COMP25111 Feedback

Questions were marked anonymously and (mostly) independently so there could be no influence of one good/poor answer from any candidate affecting judgement of other answers; the totals were only assembled when everything was complete.

Q1: How does "metadata" differ from "data"? Include a simple example. [2 marks]

Generally very well-answered. Most of the class knew what this was about and were able to provide a sensible example.

Q2: Briefly define what is revealed by the Unix command "echo $PATH". [2 marks]

This was very interesting. About half the class correctly understood this. Roughly the other half incorrectly thought that $PATH holds the full path from the root ("/") to the current working directory.

Q3: Why are mutual exclusion locks sometimes needed in operating systems code? [2 marks]

Generally very well-answered. Most of the class knew what this was about and wre able to provide a sensible example.

Q4: In Unix, lots of things appear in the files tree as "files". Give four different examples of apparent file "types" which may be found in such a filing system. [2 marks]

Generally very well-answered with sensible examples but some people thought that different examples of a particular type of file, a file which contains data, such as .php/.htm/.css were different "types" of files.

Q5: You are debugging an application by printing out messages when certain points are reached to find the position of a segmentation fault in the source code and the circumstances which preceded it. Because this takes a long time (many messages) you redirect the messages into file. Unfortunately when you look at this the file cuts off abruptly (in the middle of a message!) and you suspect the last few messages before the fault might be missing. Why might this happen and what can you do to fix it? [2 marks]

As for the "why might this happen?" part, about 2/3 of the class were able to recognise that this was to do with a buffer being partially filled but not flushed. The others made random and usually implausible guesses. Many people did not address the "what can you do to fix it?" part at all.

Q6: Write a short explanation, suitable for a student at the beginning of a Computer Science degree course, of how a computer with a single processor can appear to be "multitasking". Include some details on the possible process states, how processes maybe scheduled and how relatively slow I/O is accommodated. As a guide three or four paragraphs should be adequate. [10 marks]

Generally very well-answered and most people got full marks for this. A few people wrote very brief answers and scored low marks. A few people just listed a set of facts with no attempt at a narrative.

Q7: In one to three sentences, describe the general meaning of "exceptions" in computing. [2 marks]

Generally very well-answered.

Q8: In a Linux system a user process may be subject to exceptions, known as "signals", most of which can be trapped in the user application. Give three different examples of such exceptions. The POSIX names are not required if you can give a brief description of the cause. [3 marks]
A nicely discriminating question, mostly well answered but catching a few out with error conditions which are return values from a call rather than exceptions: 'file not found' being a favourite choice.

Note that "page fault" will not typically be echoed to a user process since it is usually handled transparenty by the paging system; only if the error is something like a *segment violation* is the user process informed. Also an array bound violation will usually not be detected at this level unless the violation is egregious enough to have caused a segmentation violation.

A small minority confused these signals with hardware-level exceptions such as interrupts but this was fewer than might have been expected.

Q9: A typical computer processor (such as an ARM or x86) will support a set of exceptions which will trap into the operating system. Give three different examples of such exceptions. [3 marks]

Generally very well-answered.

Q10: Some types of exception are "inexact". What does this mean? Suggest why some sections of Operating System code will disallow inexact exceptions. [2 marks]

Intended to be a bit more challenging there were some good answers to this. On the other hand, to address a couple of misconceptions:

* an interrupt may happen at (almost) any position in the code but - once it has happened- the position is determinable ... else return would be impossible.

* the cause of an exception can always be determined (except for rare, fault conditions such as RFI which is rather beyond this module!).

Q11-17 are related and would form a multipart question if permitted by Blackboard. Parts were marked independently.

Q11: Assuming each byte has a unique address, how many bytes are addressable in a 32-bit address space? [1 mark]

Should be trivial (and familiar). Mostly answered correctly.

Q12: In a virtual memory system, why is the memory mapped in "pages"? [2 marks]

Another straightforward question, wanting a bit more understanding. Many answers applied the 'shotgun' approach: write down lots of remembered items about pages in the hope that something hits the target. Any one good 'hit', such as space saving or convenience in swapping was given full credit. The question was still marked rather generously.

Q13: In a virtual memory system, why does each process require a unique page table/set of page tables? [2 marks]

This was generally understood.
Q14: In a 64-bit virtual memory space, why is it effectively essential to use a multi-level page table structure? Illustrate your answer with some quantitative estimates. [4 marks]

In general, poorly answered for various reasons. Most candidates were able to appreciate this was about (memory) size - and knew something about multi-level tables - but many stopped at that point. Note that the question asks for "quantitative estimates"; a good answer should give some numbers estimating the relative sizes of the table(s) concerned. Many got as far as the single-level estimate but few continued further; many of those who did exhibited little 'feel' for the range of numbers; the values concerned are large enough that any estimate should make the point. Numeracy was also a bit worrying: there were many different answers to "64 - 12 = ?" for example which should be possible even without a calculator!

Very concerning was the number of answers referring to time and "searching" tables. Page tables are ARRAYS and (assuming for a moment these are in RAM) an array look up (hardware, software, it makes no difference) is a fast *constant time* operation. This is a point which is important in lots of computing applications. There is no 'searching' involved. (Indeed, a simple sanity check ought to reveal that *one* search through the arrays of the mentioned sizes would take literally hours.)

Specifically, in this question's context, multi-level look ups would be -slower- than single level look ups (not faster) because they have more look ups to do. This is alleviated by the TLB caching the currently used translations and translating in a constant time, end-to-end. (TLBs do 'search' but are small - size need not vary with address space size - and the hardware 'search' can be parallelised giving a single cycle translation. This was not part of this question though.)

Q15: A Blu-ray disc can hold about 25 GiB of data. Given sufficient budget for buying RAM, could this all be fitted (at the same time) into the virtual address space of: i) a 32-bit computer? ii) a 64-bit computer? Full marks require a little justification, not just yes/no answers. [2 marks]

Most people got this correct, although some simply did give yes/no answers, despite the instructions in the question.

Q16: The MMAP function allows the logical mapping of a file into the addressable memory space. Assuming space is available for a particular file and the MMAP function is available, describe how data from a particular section of the file (somewhere near the middle) would be accessed using:

i) File reading operations
ii) MMAP

Exact names and syntax are not required but you should include enough description to make all the operations clear. [9 marks]

This question was intended to be fairly tough and has been marked accordingly. The intention was to view the operations from a user's perspective, successively opening the file, reading data etc. Some candidates interpreted this a bit differently and delved more into the internal operational details; in retrospect the wording of the question does not preclude this and such answers have been credited appropriately.

Note that much of this formed part of one of the practical exercises! Nevertheless, to earn most or all of the 9 marks required a number of details and these were lacking in many answers. Most (in some sense) could 'open' a file (not everyone considered the possibility of errors here); closing the file at the end was much less frequent ... and so forth.

Worth particular mention are the number of answers which explicitly treated the file purely as serial, reading and discarding data until the appropriate place was reached. The ability to 'seek' a position seems to be known only to a few, which is rather worrying.

Another worry is that some (admittedly only a few) candidates are still confused by the memory structure, believing that files are already in the addressable space. Neither does reading a file involve the creation of a separate process...

Q17: What is meant by "Lazy Loading" and "Eager Loading"? Describe the behaviour of an operating system employing Lazy Loading of pages when a large binary file is executed. [5 marks]

As well as the definitions, which should be straightforward (and were usually more-or-less described), there are a couple of subtleties in this question. The example of executing a large binary is a hint since it is likely that memory accesses will not be in a linear order and that not all the code will be used: this is typical program behaviour. Also, the mechanism by which an area of memory can be demanded should be reminiscent of paging: allocate the space but claim physical memory and fetch each page only when execution is attempted there.

This question discriminated well, in that the majority of candidates recognised the definitions and there was a range of detail and accuracy in the resulting deductions.

Q18: Explain what is meant by "DMA" in the context of I/O support. Why is DMA often employed when loading data from a hard disk? Briefly describe, in outline, all the processes which occur when setting up and loading a block of data from a disk. [5 marks]

Clearly the process is (largely) understood by most candidates. There were one or two variations on "Direct Memory Access" - "Addressing" makes sone sense; "Allocation" is not really appropriate - but, as long as the meaning was clear, such details were ignored. There is a bit less clarity in what instigates what operation: the CPU is in charge and sets up all the details, both in the DMAC and the command(s) to the disk controller; the DMAC obeys the instructions but only responds to DMA requests (similar to 'events' or interrupts) when the hardware needs servicing, sitting idle between requests.

Be cautious about dismissing devices as "slow"; whilst the latency of a disk transfer is typically large the data rate when it finally arrives can be high and DMA can move this faster than software can. It is the *overall* transfer rate which is slow, not necessarily the *peak* rate.

For some, there still seems some confusion about the role of the caches and MMU here. Whilst various architectures may be possible there is nothing much to gain by adding the cost of any memory translation to DMA: it is better to work in the physical space and should only normally be concerned with the main memory (usually DRAM). Caches are for speeding up processes on processors.

These is also confusion about interrupts with quite a few candidates having the 'disk' interrupt the processor at the start of the transfer. At this time only the CPU knows about the transfer. The sequence is:

* CPU sets up DMA parameters (device, address, length)
* CPU sets up peripheral device to begin transfer
* CPU blocks process and context switches
* Repeat
  - peripheral obtains some data and sends DMA request (similar to 'interrupt' but to DMA controller, not CPU)
  - DMA controller requests use of bus and performs transfer
* Transfer complete: DMA controller interrupts CPU & stops
* CPU unblocks waiting process