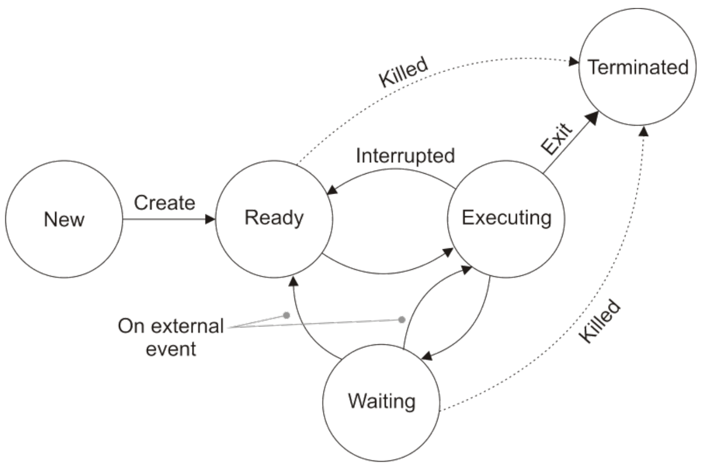
EEE6207 – Advanced Computer System – Solutions 2015/16

Q1:

(a) Using a suitable diagram, describe the range of states of process, paying particular attention to the transitions between states. Explain how each of the state transitions can arise. [5 marks]



On creation, the process is initialised and placed in the ready queue, which means it is able to run but not currently running. When selected by a scheduler, a process makes the transition to the executing state and is typically placed back into the ready state when either its time slice expires, or it voluntarily yields. A process in either the ready or executing states can be terminated (killed) by either a user action or an unrecoverable fault. In addition, an executing process can enter a waiting state in which it is awaiting some event external to the process, for example, a semaphore. The process ca exit the waiting state in one of two ways. If the process is spin locking – in other words, remaining in execution but looping on the occurrence of the event – execution can simply resume without further operation. Alternatively, when the external event occurs, the process can be placed back into the ready queue awaiting further scheduling.

In connection with the creation of child processes under the UNIX operating system, what is the copy on write mechanism? How does it work? Why is it needed? What advantages does it offer? [5 marks]

A UNIX process will typically create a child process by executing the fork system call. Under a fork, an exact clone of the parent process is created with a new, unique PID but sharing the parents resources, such as open file handles. Under classical UNIX, the memory of the parent process will be copied byte-by-byte to the newly-allocated memory of the child process. This is very inefficient the two reasons: firstly, the copying operation tends to be rather time-consuming. Secondly, almost invariably the image of the child process is overwritten with a new executable image by a call to one of the exec family of functions.

To improve the efficiency of this situation, when a child processes created the memory of the parent process is tagged as copy-on-write (COW), but an exact copy of the parent process's memory is not created as the child process. At this point, if the child process executes and performs a read operation, it is the parent's memory that is accessed – both processes share the parent's memory. If, however, either process attempts to *write* to the shared memory, this will create a conflicted memory state. Since the parent's memory is tagged as copy-on-write, the memory management unit recognises this exception and creates an independent copy of the parent's memory in the child process before proceeding.

The principal advantage of the copy-on-write mechanism is that it can produce significant speed ups in execution time.

(b) In the UNIX operating system, describe the mechanism by which a process terminates. What is the advantage of this method? How is this mechanism modified the process to be terminated is currently the parent to a child process? [6 marks]

In order to terminate process, the kernel would send a signal to the process, which would execute a signal handler function to perform a 'clean' shutdown of the process. The default signal handler would do very little other than close any open files, etc. but the mechanism allows for more elaborate shutdown operations to be carried out since a program can define its own signal handlers. At this point, the process's memory would be released and the relevant kernel data structures be purged. This mechanism would also be followed by a process terminating itself by calling the abort function – the role of the abort function will be to instruct the kernel to send the process a kill signal.

In the case were a process is a parent to a child process, the operating system would maintain a list of PIDs of child processes, most conveniently in the process control block (PCB). Hence, in addition to the above termination mechanism, and before sending a kill signal to the parent process, the kernel would examine the PCB of the parent and kill any child processes before killing the parent.

(c) What is the major advantage claimed for the Java Virtual Machine architecture? Are there any disadvantages? [4 marks]

Since the Java Virtual Machine sits between the executing Java program and the operating system, the great advantage is held to be that any Java program can execute without modification on any piece of hardware for which a Java Virtual Machine is available. Hence it is only necessary to implement a Java Virtual Machine for a new computer, not a complete set of development tools to facilitate recompilation, etc. The disadvantage of this virtual machine architecture is speed: typically, Java programs are not interpreted (like Basic) but produce byte code output. Java programs thus run faster than purely interpreted code, but not as fast as native code.

Q2:
(a) Modern operating systems typically improve the performance of their filesystems by caching blocks of data in memory rather than making processes wait while data written to the physical disk; the cached blocks are typically written to the disk at some later point by kernel threads, allowing the write operation to return the calling program. Although caching improves speed of operation, it carries a major disadvantage. Explain why? [4 marks]

The major disadvantage caching is lack of resilience in the event of a computer crash. Clearly cached data which have not yet been written to physical disk will be lost during a crash. The situation is typically more involved than this, however, since the optimisation of the file system typically writes cached blocks in non-sequential order to minimise the disk write times. As a consequence, if a crash occurs during the physical write phase, the file is not merely incomplete but, potentially, seriously corrupted.

Describe the basic processing steps involved in a redo journalling file system. How does a journalling file system overcome the disadvantage mentioned above? What is the great disadvantage of journalling? [4 marks]

The basic problem with crash resilience outlined above is that the data written to a file on the can become inconsistent. A very similar problem presents in database systems and hence journalling is borrowed from databases. Basic redo journalling comprises three phases:

1. Writing the transactions (meta data and user data) to a special *journal file*. The transactions and then said to have been *committed* to the system.

2. Perform the actual transaction file system.

3. Delete the just-completed transactions from the journal file.

A computer crash during phase 1 will lose the current transactions but any subsequent recovery phase can recognise the loss of transactions due to an incomplete journal file. A crash during phase 2 can be detected because the transactions are still present in the journal file. Thus any recovery can redo the transactions, which gives the technique its name. Finally, a crash during phase 3 results in a corrupted journal file from which we can infer that the transactions have been sent to physical disk but not removed from the journal file; in this eventuality there is no corrective action necessary.

The great disadvantage of journalling is that it involves many more disk writes to the journal file. These impose an overhead on the system and lead to reduced performance, which claws back some, but not all, of the performance gains from using caching in the first place.

(b) In the context of real-time operating systems, describe what is meant by the two terms:
1. Interrupting latency
2. Dispatch latency

[4 marks]

Interrupting latency is the delay between an interrupting signalled in the CPUs starting to execute the interrupt service routine to handle that interrupt.

Dispatch latency, on the other hand, is the delay in pre-empting one process and starting a new process.

Explain why allowing pre-emption of the kernel can improve the performance of real-time systems. [4 marks]

In conventional operating systems, it was once the practice to disable all interrupts when a process entered a system call, only re-enabling interrupts when the process exited the system call. Such a system has a very large and unpredictable interrupt latency and is generally undesirable for real-time operation. In order to reduce interrupt latency, it is desirable to allow system calls to be interrupted although this has to be done with great care to prevent corruption of the kernel. One approach is to define *safe pre-emption points* in the kernel code to reduce the time between safe opportunities to interrupt a system call. Further reduction in the interrupt latency can be achieved by allowing the kernel to be *fully pre-emptible* but this comes at the cost of considerable increase in the kernel complexity. Additionally, low priority processes which can be interrupted need to be able to lock kernel data structures. This is generally not a problem except in the case of priority inversion.

What do you understand by *priority inversion*? How can priority inversion be prevented? [4 marks]

In priority inversion, a low priority process holds kernel data structure is required by a high priority process. Normally, since the high priority process would be blocked on access to the data structure, the low priority process would be allowed to execute long enough to release this locked structure. A problem does arise, however, if an intermediate-priority process is introduced. Since this intermediate-priority process has a higher priority than the low priority process (which is blocking the high priority process), low priority process is prevented from running. In effect, intermediate-priority process is preventing the high priority process from running.

Priority inversion can be prevented by recognising that the high priority process is blocked on the low priority process, and temporarily assigning the low priority process a high priority to allow it to release the blocking resource.

3. *A pipelined system appears as shown in **Figure 3**.*

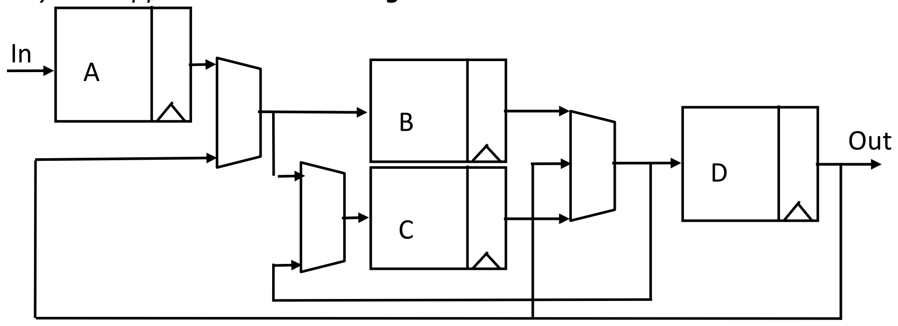


Figure 3: Pipelined System

The function of A, B and D are fixed. However, C is flexible: in particular, it can be programmed – on-the-fly– to also execute a no-operation, operation B, and operation E.

a. *The pipeline is required to execute the following sequence of operations:*

A→B→C→D→E→D

i) *Produce a reservation table to identify how the sequence of operations will be scheduled*

Recognising that the block labelled C must be also used to produce the function E we can show that:

Time	A	B	NoOp/B/C/E	D
0	D1			
1	D2	D1		
2		D2	D1 (C)	
3			D2 (C)	D1
4	D3		D1 (E)	D2
5	D4	D3	D2 (E)	D1
6		D4	D3 (C)	D2
7			D4 (C)	D3
8	D5		D3 (E)	D4
9	D6	D5	D4 (E)	D3

(4)

ii) *Identify the throughput and latency.*

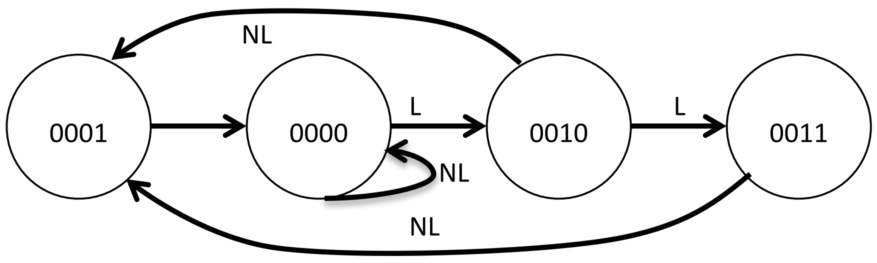
The throughput is 2 datum every 4 clock cycles and the latency is 6 clock cycles. **(2)**

iii) *Work out the utilisation of each processing block.*

Block A and B are utilized 50%. Blocks C and D are utilized 100% **(2)**

iv) *Write down a collision vector for this operation and draw the corresponding state transition diagram*

Only blocks C and D are reused and, therefore, contribute to the collision vector. In both cases, there is a gap of 1 between use and reuse. Consequently, the collision vector is 0010. The corresponding state diagram is, therefore, quite simple



(4)

b. *The operation is changed to:* **(8)**

A→B→D→B→D→D
Your objective is to get as much performance out of the system. How would you do this?

In this case you should recognize that if we just feed the output of D back onto itself then we would end up with a reservation table as follows:

Time	A	B	Nop/B/C/E	D
0	D1			
1		D1		
2				D1
3		D1		
4	D2			D1
5		D2		D1
6				D2
7		D2		
8				D2
9				D2

And this gives a throughput of 1 datum every 4 clock cycles (1/4).

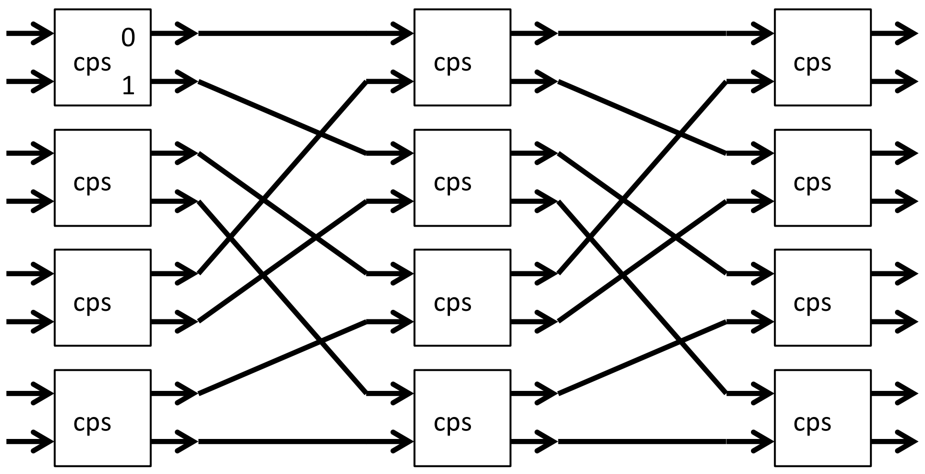
However, if we take note of the fact that C can be reprogrammed to be a NOP then we get:

Time	A	B	NoOp/B/C/E	D
0	D1			
1	D2	D1		
2		D2		D1
3		D1		D2
4		D2		D1
5			D1 (NoOp)	D2
6	D3		D2 (NoOp)	D1
7	D4	D3		D2
8		D4		D3
9		D3		D4
10		D4		D3

And, in this case, we get 2 datum every 6 clock cycles (1/3). This is an improvement of 4/3 = 1.33x the performance of the naïve approach

4. **a.** *A Delta network of size 2^3 x 2^3 is to be constructed.*

i) *Draw this network, identifying its important features.*



It is an a^n x b^n delta network with a^n inputs and b^n outputs (8 and 8 in this case). It is made up from a x b (=2 x 2) cps and has n (3) columns of cps separated by an a out of … shuffle network (both 2 out of 8 shuffles). **(8)**

ii) *Demonstrate that the network exhibits relative and absolute addressing when routing data to a particular output.*

If we represent the output address in bas b (2 in this case) we can number the outputs as 000, 001, 010, 011, 100, etc (from the top output down). If we number the outputs for each cps as shown in the top lh cps. The this base b number is the absolute address of the output and also the routing information. From any input, each bit (reading from the left) identifies which output the data must emerge from each cps it passes through. So , working from the left, and picking any input, to get to output 5, the data must emerge from output 1 of the cps in the left column, output 0 of the cps in the middle column and output 1 of the cps in the right column. **(2)**

iii) *Calculate the areal efficiency (in terms of hardware) of this network – compared with a cross-point switch with the same number of inputs and outputs.*

The basic unit of complexity is the switch in a cps. An n x m cps has n*m switches. Consequently, an 8 x 8 cps (which would equate to this network) contains 64 switches. The delta network, above has 12, 2 x 2 cps and, therefore, contains 12*2*2 switches = 48 switches. The delta network, on this basis occupies 75% of the area of the corresponding cps (offset, probably, because of the lack of regularity of connections). **(2)**

b. **i)** *If a network, such as the one in part **a.** is to be used to connect processors to memory then what conditions might this impose on the network?*

To act as a connection between a set of processors and memories then the switch must be extremely high performance; it must be capable of routing a large address, wide data bus, and control information from the processor to the memory and it must also be able to route data back from the memory to the processor. It would also, typically, be capable of transferring blocks of data (cache lines) associated with a single memory reference. **(4)**

ii) *What is a split-transaction bus and what problem does it set out to solve?*

A split-transaction bus is generally used where scarce routing resources connect processors to memories. The problem is during a memory read. Memory is typically relatively slow. A transfer consists of an address phase (during which the address to the memory is set up) and then a data phase (where the data is transferred). In the case of a write, the data can be sent directly after the address and buffered at the memory until the write is completed (during which time the processor can continue). During a read, the processor has to wait for the memory to be read before the data can be returned. During this time, the processor would, typically, keep control of the routing resource and wait for the data. This creates a problem because this scarce routing resource is unavailable for other accesses during this period. The solution is to split the address and data phases during a read. The address is sent and the routing resource is relinquished – allowing the routing to be used by other processors. When the data is ready, the memory has to gain access to the routing resource (as a master rather than a slave) and transfer the data back to the processor that requested it in the first place. **(4)**