DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING

Spring Semester 2014-15 (2 hours)

Answers to EEE6207 (Advanced Computer Systems), Questions 1…4

1. a. *Distinguish between Deadlock:*

i. *Prevention*
ii. *Avoidance*
iii. *Recovery*
iv. *Ignorance*

Deadlock prevention is a static approach that imposes rules on the system such that deadlock – even *potential* deadlock – can never occur. Clearly eliminating any of the four conditions (Mutual Exclusion, Hold and Wait, No Preemption, Circularity) which characterise deadlock will achieve this goal.

Deadlock avoidance enforces policies to avoid deadlock at runtime. To implement deadlock avoidance, each process must declare in advance the maximum number of each resource it will request during its whole execution. If a process wishes to convert its claim into an actual resource request, the system analyses whether granting that request will put the systems into an unsafe state. If it will then the request is deferred and reconsidered at some later time.

Deadlock Recovery allows resources to be allocated on a first-come-first served basis which might produce a deadlocked state. Although we will no longer be needlessly delaying processes which will not deadlock, we need to deal with those processes which do deadlock. In short, we need a means of detecting when deadlock has occurred and then for recovering from the deadlock.

Ignorance just ignores the problem and then leaves it for the user to deal with the situation when it arises – usually by killing processes and trying again. **(6)**

b. Explain what the graphical items, shown in **Figure 1A**, represent with reference to *Resource Allocation Graphs* (RAG).

(a) **(b)**

Figure 1A: Elements from Resource Allocation Graphs

(a) represents a resource that can be shared. The number of dots relates to the number of instances of the resource that can be shared (in this case three instances).
(b) Represents one of a set of process that, within the context of a RAG, will be vying for access to the set of resources shared across the set of processes. **(2)**

c. *Explain the meaning of the three kinds of arcs that can be used to connect elements **(a)** and **(b)** from **Figure 1**.*

A directed arc (represented by a solid line) beginning at a process and ending at a particular resource represents an unrequited request for access to the resource.

A directed arc (represented by a solid line) beginning at a a dot within a resource and ending at a particular process represents an instance of the resource that granted to the process.

A directed arc (represented by a dotted/dashed line) beginning at a process and ending at a particular resource represents a *potential* request for access to the resource that the process may make.

(3)

d. *Describe the system state that the RAG in **Figure 1B** represents.*

*Is the system in **Figure 1B** deadlocked? Show why this is the case (by reducing the graph).*

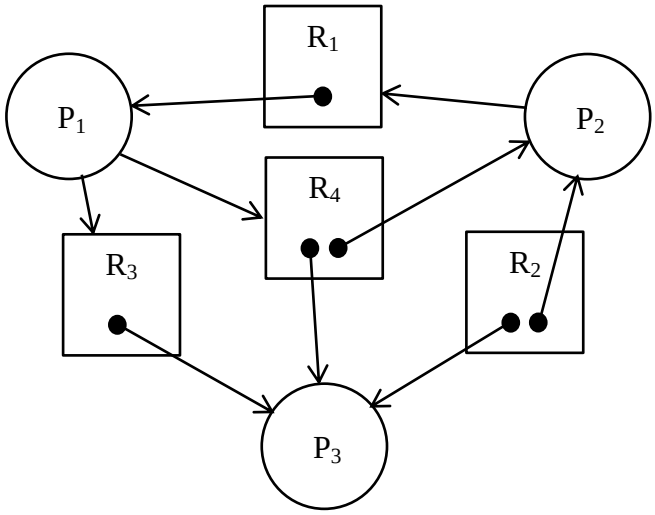


Figure 1B: Resource Allocation Graph

P_1 holds the only instance of R_1 and is requesting instances of R_3 and R_4.

P_2 holds one of the two instances of both R_2 and R_4 and is requesting access to the only instance of R_1.

P_3 holds the only instance of R_3 and and instance of both R_2 and R_4.

No. The RAG does not represent a deadlocked system, as follows:

P_3 holds all the resources it needs and is waiting for nothing. Therefore it can run to completion, freeing instances of R_2, R_3 and R_4.

P_1 gains access to R_3 and R_4. P_1 can then run to completion, freeing instances of R_1, R_3 and R_4.

P_2 can now gain access to R_1. It now has all the resources it needs and runs to completion.

(5)

e. *A dotted arc is added to the RAG in **Figure 1B**, pointing from P_3 to R_1. Comment on what this might mean about whether the system is Deadlocked.*

The dotted arc implies that P_3 may wish access to R_1. If this is converted into a claim then P_3 cannot run to completion because P_1 holds the only instance of R_1. In this case, we could trace a directed set of arrows $P_3 \rightarrow R_1 \rightarrow P_1 \rightarrow R_3 \rightarrow P_3$. That is a circular set of dependencies. In this case, the system would be deadlocked.

(4)

2. *A pipelined system is as shown in* ***Figure 2****.*

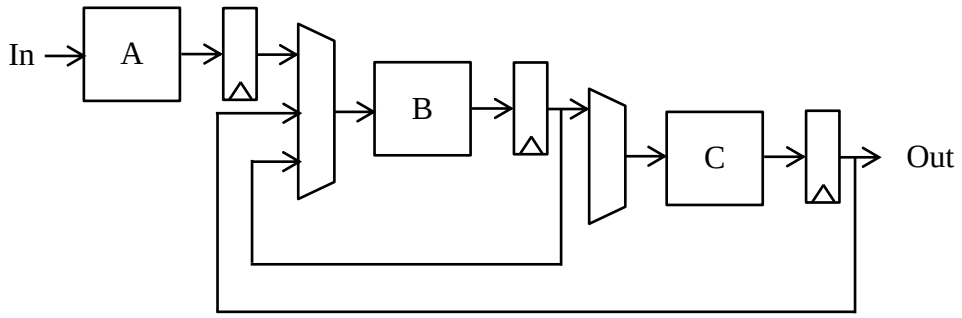


Figure 2: A Processing Pipeline

The sequence of processing is as follows:

$$In \rightarrow A \rightarrow B \rightarrow C \rightarrow B \rightarrow B \rightarrow C \rightarrow Out$$

a. *Draw the reservation table for this sequence of processing*

Time	A	B	C
0	D_1		
1		D_1	
2			D_1
3		D_1	
4	D_2	D_1	
5		D_2	D_1
6			D_2
7		D_2	
8	D_3	D_2	
9		D_3	D_2
10			D_3
11		D_3	
12	D_4	D_3	
13		D_4	D_3

(6)

b. *Calculate the throughput and the utilisation of the processors.*

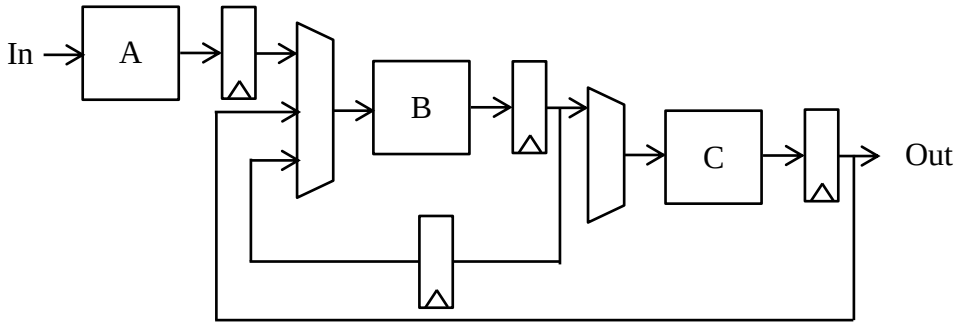
The throughput is ¼ = 0.25 datum per clock

The utilisation of the processors is A=25%, B=75%, C=50% **(2)**

c. *You recognise that there is a simple way to improve performance (without replicating any of the functional blocks).*

i. *Identify this simple way of improving performance.*

Clearly, the repeated use of B without gaps creates congestion that results in poor throughput and utilisation. If we insert a pipeline delay in the feedback between the register at the output of B and the multiplexer at the input of B then this should improve things.



(4)

ii. *Calculate the improved throughput and utilisation.*

Time	A	B	C
0	D_1		
1	D_2	D_1	
2		D_2	D_1
3		D_1	D_2
4		D_2	
5		D_1	
6	D_3	D_2	D_1
7	D_4	D_3	D_2
8		D_4	D_3
9		D_3	D_4
10		D_4	
11		D_3	
12		D_4	D_3
13			D_4

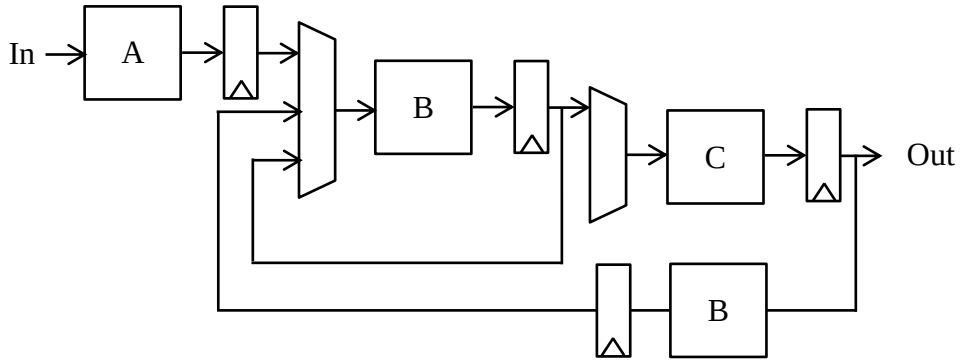
This gives 2 datum every 6 clock-cycles = 0.33 datum/clock. The utilisation of the processors is A=33%, B=100%, C=66%. At this point, B is saturated and there are no more easy gains to be made.

(3)

d. *You need to improve the performance to 0.5 datum / clock cycle. How would you do this?*

In order to improve performance further, to 0.5 datum/clock, given that B is saturated, you need to add another instance of B. So, for example, add another instance of B (B1) fed from the register at the output of C and leading back to the multiplexer at the input of B. Such that the sequence of processing is now:

In → A → B → C → B1 → B → C → Out

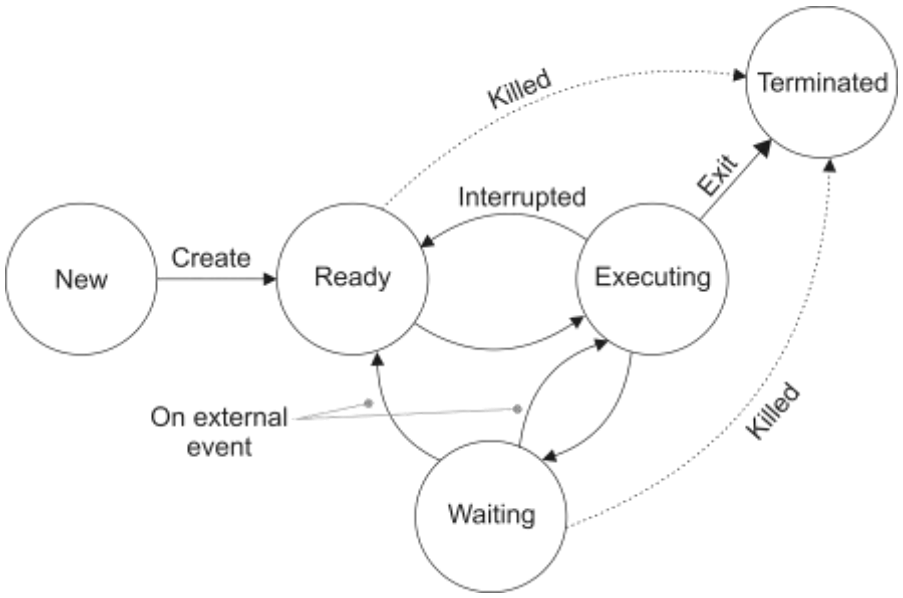


In this case, the reservation table will be:

Time	A	B	B1	C
0	D_1			
1	D_2	D_1		
2	D_3	D_2		D_1
3		D_3	D_1	D_2
4		D_1	D_2	D_3
5		D_2	D_3	D_1
6	D_4	D_3		D_2
7	D_5	D_4		D_3
8	D_6	D_5		D_4
9		D_6	D_4	D_5
10		D_4	D_5	D_6
11		D_5	D_6	D_4
12		D_6		D_5
13				D_6

and the throughput will be 0.5 datum/clock. **(5)**

3. **a.** *Identify the states that a process can be in and the transitions between the various states (e.g. what conditions can give rise to a transition).*



(3)

b. *Distinguish between a blocking and non-blocking call (made, for example, for the purposes of inter-process communication).*

A blocking call is such that when, for example, the communication cannot proceed the process making the call enters a Waiting state and is suspended. The process is only returned to the Ready state when the communication can go ahead.

In this case, a non-blocking call would return immediately with an exit code indicating that the communication could not proceed, leaving it to the process to explicitly decide what to do. **(2)**

c. *Describe the elements of a typical semaphore.*

A semaphore consists of a variable, *S*, which is owned by the kernel, and two functions that act upon *S*: *wait()* and *signal()*. Code representing the behaviour of the functions is:

```
wait() {
  while (S <= 0) ;
  S—;
}


signal() {
  S++;
}
```

(3)

d. ***i.*** *What problems are encountered in implementing a semaphore practically?*

The basic problems with a semaphore are:
indivisibility – ensuring that once a process starts dealing with the semaphore, other processes are blocked from doing so – to prevent misbehaviour;
fair access – which of the number of processes that are nominally waiting for *S* to be > 0 see it in which order;
performance – whilst processes are nominally 'spinlocking' on the *while* statement in *wait()* they are consuming processing power needlessly. **(2)**

ii. *Describe a practical implementation of a semaphore that addresses these problems (in particular, identifying hardware mechanisms that must exist).*

We maintain (within the kernel) a queue of processes which it is currently blocking (in addition to the counting variable, *S*). When a process executes a *wait()* and finds itself blocked, rather than repeatedly checking *S* it:

- adds the process's PID to the semaphore's queue of blocked processes
- Changes the process's state to WAITING. (Recall that the dispatcher will only consider processes in the `READY` state for running.)
- Yield to the process dispatcher

When the semaphore is subsequently signalled, the *signal()* function may increment the counting variable, *S*, but if the queue is not empty, instead, it removes the blocked process' PID from the head of the queue, changes its state back to *READY*, thereby making it a candidate to run.

The functions still have to access, indivisibly, *S* and the queue – but for a very short time. To enable this, access to these is controlled by a simple *meta-semaphore* which, in turn relies on a hardware instruction (TEST&SET or EXCHANGE) which is inherently indivisible in a uni-processor system. **(4)**

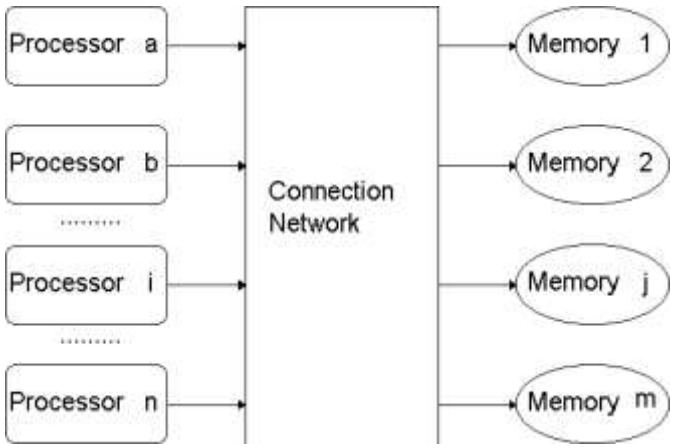
e. *Two processes must communicate with each other. The first process, **A**, initiates the process by sending data to the second process, **B**. **B** then responds by sending data back to **A**. The process repeats. Show how this communication should be performed (either using semaphores and shared memory or other inter-process communication constructs that might be available in typical operating systems).*

Process A	Process B
S1=0, S2=0	
Repeat { *…* *write Data from A to B* *signal(S1)* *…* *wait(S2)* *read Data B to A* *}*	*Repeat {* *…* *wait(S1)* *read Data A to B* *…* *write Data B to A* *signal(S2)* *}*

(6)

4. **a.** *A set of processors is connected by a cross-point switch to a number of disjoint memory modules:*

i. *Draw a schematic to represent this system organisation.*



(2)

ii. *What is the principal feature of the cross-point switch that helps to improve performance in this system organisation.*

It is non-blocking. That is, the routing/connection of an input to an output does not preclude the connection of any other input to any other output. **(2)**

b. *The probability, P_A, that a memory access issued by a processor in part **a.** goes ahead without being blocked is:*

$$P_A = \frac{M}{NRT_{acc}}\left(1 - \left(1 - \frac{RT_{acc}}{M}\right)^N\right)$$

What do each of the terms in this equation mean and what assumptions are used to arrive at this equation.

N is the number of processors, M is the number of memory units, T_{acc} is the basic memory access time across the switch, R is the rate at which a processor is issuing memory accesses across the switch.

The assumptions are: that the memory traffic generated by each processor is independent of the others; the probability with which a processor issues a memory access to each memory unit is the same; if two or more processors issue a memory access to a particular memory unit in one time slot, one will be granted an the others will be blocked and then need to reissue the memory access in the next time slot; the rate at which memory accesses are generated does not depend on the failure of one or more accesses to be granted. **(4)**

c. *If a particular memory access through the cross-point switch is blocked (by another memory access) then the processor must try again (in the next time-slot) until it successfully completes the transfer of data. In this case, show that the effective access time (using relevant terms from the equation above) is:*

$$accesstime = \frac{T_{acc}}{P_A}$$

Every access requires one access, which costs T_{acc}. However, a proportion of these accesses ($1-P_A$) will fail resulting an additional access time. Of this proportion that require a subsequent access, ($1-P_A$) of these will fail. That is, the proportion that will require a 3rd access is ($1-P_A$)($1-P_A$), and so on. Consequently,

the average (or effective) access time is:

$accesstime = T_{acc}+(1-P_A)T_{acc}+(1-P_A)^2T_{acc}+(1-P_A)^3T_{acc}+(1-P_A)^4T_{acc}+\ldots$

And this is $accesstime = \frac{T_{acc}}{1-(1-P_A)} = \frac{T_{acc}}{P_A}$ **(4)**

d. *Eight processors must be connected via a cross-point switch to a set of memory blocks (the number of which blocks is to be determined). Each processor is generating 10^9 memory accesses per second. However, there is an internal cache in each processor such that the overall miss rate on a cache is 1%. If a processor does miss, the basic time to complete a memory operation through the cross-point switch to the memories is 50ns.*

Derive an expression to identify the relative performance of systems with different numbers of disjoint blocks of memory (hint: consider how long it takes to complete the number of accesses that should be completed within one second).

Simplifying the expression the probability of an access across the network being accepted based on what we know:

There are 10^9 accesses per second but only 1% of them go across the network and so R=0.01x10^9. In this case, T_{acc} is 50ns. So, RT_{acc} = 0.5

$$P_A = \frac{M}{8 \times 0.5}\left(1-\left(1-\frac{0.5}{M}\right)^8\right) = \frac{M}{4}\left(1-\left(1-\frac{0.5}{M}\right)^8\right)$$

Consequently, the access time for any access across the network with be:

$$accesstime = \frac{50 \times 10^{-9}}{\frac{M}{4}\left(1-\left(1-\frac{0.5}{M}\right)^8\right)} = \frac{200 \times 10^{-9}}{M\left(1-\left(1-\frac{0.5}{M}\right)^8\right)}s$$

The total number of nominal accesses within 1 second that need to go across the network is 0.01 x 10^9 and so the total time accessing across the network is:

$$externaltime = \frac{0.01 \times 10^9 200 \times 10^{-9}}{M\left(1-\left(1-\frac{0.5}{M}\right)^8\right)} = \frac{2}{M\left(1-\left(1-\frac{0.5}{M}\right)^8\right)}s$$

So (neglecting the effect that this has on R), in every second 99% of the accesses are satisfied by the cache taking 0.99 seconds and so the total time would be:

$$totaltime = 0.99 + \frac{2}{M\left(1-\left(1-\frac{0.5}{M}\right)^8\right)}s$$

(8)

NLS/NJP