SYSC 4001 Operating Systems
Solutions to Sample Midterm 1

Question 1. Bakery Algorithm (20 marks)

The *bakery algorithm* provides mutual exclusion to a number of processes, based on the practice observed in bakeries and other shops in which every customer receives a numbered ticket on arrival, allowing each to be served in turn. The algorithm is as follows:

```
Process Pi:
repeat
        choosing[i]:=true;
        number[i]:=max(number[0]..number[n-1])+1;
        choosing[i]:=false;
        for j:=0 to n-1 do {
                while (choosing[j]) {};
                while (number[j]!=0 and (number[j],j)<(number[i],i)){};
        }
        CS
        number[i]:=0;
        RS
forever
```

The arrays *choosing* and *number* are initialized to *false* and 0, respectively. The *i*th element of each array may be read and written by process *i* but only read by other processes. The notation (a,b) < (c,d) is defined as:

(a < c) or (a = c and b < d)

a) Describe the algorithm in words

Answer (5 marks):
When a process wishes to enter its critical section, it is assigned a ticket number. The ticket number assigned is calculated by adding one to the largest of the ticket numbers currently held by the processes waiting to enter their critical section and the process already in the critical section. The process with the smallest ticket number has the highest precedence for entering its critical section. In case more than one process receives the same ticket number, the process with the smallest numerical name enters its critical section. When a process exits its critical section, it resets its ticket number to zero.

b) Show that this algorithm avoids deadlocks

Answer (5 marks):
If each process is assigned a unique process number, then there is a unique, strict ordering of processes at all times. In case two processes try to enter their critical section at the same time and obtain the same ticket number, they are ordered by their unique numerical name, so a strict total ordering of processes still exists. Therefore, it is well defined what process will enter a critical section next, deadlock cannot occur.

c) Show that it enforces mutual exclusion

Answer (10 marks):
This is somewhat more involved, and the following is the formal proof. For the sake of the midterm, plausibility arguments that capture the same line of reasoning are sufficient.

To demonstrate mutual exclusion, we first need to prove the following lemma: if Pi is in its critical section, and Pk has calculated its number[k] and is attempting to enter its critical section, then the following relationship holds: (number[i],i) < number[k],k).

To prove this lemma, define the following times:

Tw1: Pi reads choosing[k] for the last time, for j=k, in its first wait, so we have choosing[k] = false at Tw1
Tw2: Pi begins its final execution, for j=k, of the second while loop. We therefore have Tw1 < Tw2.
Tk1: Pk enters the beginning of the repeat loop
Tk2: Pk finishes calculating number[k]
Tk3: Pk sets choosing[k] to false. We have Tk1 < Tk2 < Tk3

Since at Tw1, choosing[k] = false, we have either Tw1 < Tk1 or Tk3 < Tw1.

a) In the first case, we have number[i] < number[k], since Pi was assigned its number prior to Pk; this satisfies the condition of the lemma.
b) In the second case, we have Tk2 < Tk3 < Tw1 < Tw2, and therefore Tk2 < Tw2. This means that at Tw2, Pi has read the current value of number[k]. Moreover, as Tw2 is the moment at which the final execution of the second while for j = k takes place, we have (number[i],i) < (number[k],k), which completes the proof of the lemma.

It is now easy to show that mutual exclusion is enforced. Assume that Pi is in its critical section and Pk is attempting to enter its critical section. Pk will be unable to enter its critical section as it will find number[I] ≠ 0 and (number[i],i) < (number[k],k).

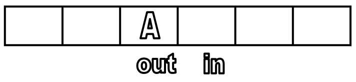
Question 2. Producer/Consumer Problem with Bounded Buffer (10 marks)

The solution of the bounded buffer problem with finite buffer presented in class allows only at most n-1 entries in the buffer. For your information, the code is included below again:

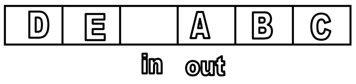
```
Initialization: S.count:=1;
                N.count:=0;
                E.count:=k;

Producer:       Consumer:        append(v):
Repeat          repeat           b[in]:=v;
  Produce v;      wait(N);       in:=(in+1)
  Wait(E);        wait(S);          mod k;
  Wait(S);        w:=take();
  Append(v);      signal(S);     take():
  Signal(S);      signal(E);     w:=b[out];
  Signal(N);      consume(w);    out:=(out+1)
Forever         forever             mod k;
                                 return w;
```

a) Explain why this is the case.

Answer (5 marks):
If the buffer is allowed to contain n entries, then the problem is to distinguish an empty buffer from a full one. Consider a buffer with six slots, with only one entry, as follows:



Then, when the one element is removed, out = in. Now suppose that the buffer is one element shy of being full:



Here, out = in + 1. But then, when an element is added, in is incremented by 1 and out = in, the same as when the buffer is completely empty.

b) Modify the algorithm to remedy this deficiency.

Answer (5 marks):
You could use an auxiliary variable, count, which is incremented and decremented appropriately. Then, the buffer is completely empty for in = out AND count = 0, and completely full for in = out AND count = n.

NOTE: I took the question and solution from the textbook. It turns out that there is indeed no problem, as some students discovered: if in=out, the correct operation is ensured by the semaphores. For an empty buffer, E will prevent the Consumer from taking out an element, for a full buffer, N will prevent the Producer from putting more in. Just goes to show that using semaphores and reasoning about them is not easy……. ☺

Question 3. Banker's Algorithm (10 marks)

a) Consider a system with a total of 150 units of memory, allocated to three processes as shown:

Process	Max	Hold
1	70	45
2	60	40
3	60	15

Apply the banker's algorithm to determine whether it would be safe to grant each of the following requests. If yes, indicate a sequence of terminations that could be guaranteed possible. If no, show the reduction of the resulting allocation table and identify the processes involved in a deadlock:

i) A fourth process arrives, with a maximum memory need of 60 and an initial need of 25 units.

Solution (4 marks):
Creating the process would result in the state:

Process	Max	Hold	Claim	Free
1	70	45	25	25
2	60	40	20	
3	60	15	45	
4	60	25	35	

There is sufficient free memory to guarantee the termination of either P1 or P2. After that, the remaining three jobs can be completed in any order.

ii) A fourth process arrives, with a maximum memory need of 60 and an initial need of 35 units.

Solution (4 marks):
Creating the process would result in the trivially unsafe state:

Process	Max	Hold	Claim	Free
1	70	45	25	15
2	60	40	20	
3	60	15	45	
4	60	35	25	

b) Evaluate the banker's algorithm for its usefulness in real life.

Solution (2 marks):
It is unrealistic: don't know max demands in advance, number of processes can change over time, number of resources can change over time (something can break).