CS3SH3 Test 2 Winter 2003

Name ____________________ Student No. ____________________

No aids allowed. Answer all questions on test paper. Use backs of sheets if necessary.

Total Marks: 40

1. Recall the *Bakery Algorithm* for n-process synchronization presented in class. The common data structures are: `boolean choosing[n]` and `int number[n]`, and the structure of process P_i (for $0 \le i \le n-1$) is given by:

```
1      do {
2         choosing[i]=true;
3         number[i]=max{mumber[0],number[1],...,number[n-1]}+1;
4         choosing[i]=false;
5         for (j=0; j<n; j++) {
6             while (choosing[j]);
7             while ((number[j]!=0)&&((number[j],j)<(number[i],i)));
8         }
9             -- CRITICAL SECTION --
10        number[i]=0;
11            -- REMINDER SECTION --
12        } while(1);
```

[10] (a) Recall that $(a,b) < (c,d)$ if $a < c$ or if $a = c$ and $b < d$. Show that if process P_i is in its critical section and P_k ($k \neq i$) has *already* chosen its `number[k]!=0`, then `(number[i],i)<(number[k],k)`.

If P_k gets its number *while* P_i is already in its critical section, then from line 3 we can see that `number[k]>number[i]`, and so `(number[i],i)<(number[k],k)`.

If, on the other hand, P_k gets its number before P_i enters its critical section, then that means that they have been competing to enter their respective critical sections, and P_i won. Since in line 6 P_i waits for P_k to get its number (and vice-versa), it means that after P_k chose its number, P_i completed its for-loop and entered its critical section, so the condition `((number[k],k)<(number[i],i))` must have failed. If `((number[k],k)<(number[i],i))` fails, and since we know that $k \neq i$, it follows that `((number[k],k)>(number[i],i))` holds as desired.

[5] (b) Show that the condition of *mutual exclusion* is observed. (Hint: use (a).)

Suppose that P_i is in its critical section, and P_k attempts to enter its critical section.

When P_k executes line 7 with $j = i$, it finds that the first condition holds (that is `number[i]!=0`) since P_i has a number as it is in its critical section, and the second condition (`(number[i],i)<(number[k],k)`) holds by (a).

Thus, P_k will continue in line 7 with $j = i$ until P_i leaves its critical section and resets its number to zero.

[5] (c) Show that *progress* and *bounded-waiting* are also preserved.

Progress is ensured by the fact that when a process enters its reminder section, it has set its number to zero, and hence it does not participate in the competition to enter critical sections.

Bounded-waiting is ensured by the fact that no process has to wait for more than n other processes to go through their critical sections. Once a process exits its critical section, it sets its number to zero, so next time it competes to enter its critical section, its number will be bigger than the number of the process that had to wait for him.

2. Recall *Banker's Algorithm* for deadlock avoidance where resources have several instances. Consider the following snapshot of a system:

	Allocation	Max	Available
	ABCD	ABCD	ABCD
P_0	0,0,1,2	0,0,1,2	1,5,2,0
P_1	1,0,0,0	1,7,5,0	
P_2	1,3,5,4	2,3,5,6	
P_3	0,6,3,2	0,6,5,2	
P_4	0,0,1,4	0,6,5,6	

Here P_i's are processes, and A,B,C,D are resource types.

[5] (a) Explain what are *Allocation, Max* and *Available.*

Allocation denotes how many instances of a given resource are currently allocated to a given process.

Max denotes the maximum demand that a given process may make on a resource type.

Available indicates the number of available resources of each type.

[5] (b) What is the content of the matrix *Need*? Explain what *Need* is.

Need is simply Max minus Allocation, that is, the upper bound on the remaining resource need of each process. In the above case it is given by:

$$\text{Need} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 7 & 5 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 6 & 4 & 2 \end{bmatrix}$$

[10] (c) Is the system in a safe state ? Describe briefly the algorithm that you are using to establish this.

To check this we use the **safety algorithm** on page 258 in the textbook (giving this algorithm is part of the answer, worth 1/2 of the mark):

First: Work = Available = $[1, 5, 2, 0]$, and Finish = $[F, F, F, F, F]$ ($F = \mathit{false}$, and $T = \mathit{true}$).

1. Is there an $0 \leq i \leq 4$ such that Finish$[i] = F$ and Need$_i \leq$ Work ? Yes, $i = 0$. So Work = Work + Allocation$_0$ = $[1, 5, 2, 0] + [0, 0, 1, 2] = [1, 5, 3, 2]$ and Finish$[0] = T$.

2. $i = 2$. Work = Work + Allocation$_2$ = $[1, 5, 3, 2] + [1, 3, 5, 4] = [2, 8, 8, 6]$ and Finish$[2] = T$.

3. $i = 1$. Work = Work + Allocation$_1$ = $[2, 8, 8, 6] + [1, 0, 0, 0] = [3, 8, 8, 6]$ and Finish$[1] = T$.

4. $i = 3$. Work = Work + Allocation$_3$ = $[3, 8, 8, 6] + [0, 6, 3, 2] = [3, 14, 11, 8]$ and Finish$[3] = T$.

5. $i = 4$. Work = Work + Allocation$_4$ = $[3, 14, 11, 8] + [0, 0, 1, 4] = [3, 14, 12, 12]$ and Finish$[4] = T$.

Now Finish = $[T, T, T, T, T]$, so the system **<u>is</u>** in a safe state.