University of Newcastle
School of Electrical Engineering and Computing

COMP2240 - Operating Systems
Workshop 6
Topics: Concurrency: Deadlock and Starvation

1. Consider the following software solution to the mutual exclusion problem. It happens to be incorrect. Find a condition under which this solution fails.

```
var     blocked: array[0..1] of boolean;
        turn: 0..1;
procedure P(id: integer);
begin
        repeat
                blocked[id] := true;
                while turn <> id do
                        begin
                                while blocked[1 - id] do
                                        {nothing};
                                turn := id;
                        end;
                <critical section>
                blocked[id] := false;
                <remainder>;
        until false
end;
```

```
begin
        blocked[0] := false;
        blocked[1] ;= false;
        turn := 0;
        parbegin
                P(0); P(1);
        parend
end.
```

2. At an instant, the resource allocation state in a system is as follows:
4 processes P1-P4
4 resource types: R1-R4
R1 (5 instances), R2 (3 instances), R3 (3 instances), R4 (3 instances)
Snapshot at time T_0:

	Allocation				Request				Available			
	R1	**R2**	**R3**	**R4**	**R1**	**R2**	**R3**	**R4**	**R1**	**R2**	**R3**	**R4**
P1	0	0	1	0	2	0	0	2	2	1	1	2
P2	2	0	0	1	1	3	0	1				
P3	0	1	1	0	2	1	1	0				
P4	1	1	0	0	4	0	3	1				

Run the deadlock detection algorithm and test whether the system is deadlocked or not. If it is, identify the process that are deadlocked.

3. In the code below, three processes are competing for six resources labeled A to F.

a) Using a resource allocation graph (Figures 6.5 and 6.6), show the possibility of a deadlock in this implementation.
b) Modify the order of some of the get requests to prevent the possibility of any deadlock. You cannot move requests across procedures, only change the order inside each

procedure. Use a resource allocation graph to justify your answer.

```
void P0()
{
  while (true) {
    get(A);
    get(B);
    get(C);
    // critical region:
    // use A, B, C
    release(A);
    release(B);
    release(C);
  }
}
```

```
void P1()
{
  while (true) {
    get(D);
    get(E);
    get(B);
    // critical region:
    // use D, E, B
    release(D);
    release(E);
    release(B);
  }
}
```

```
void P2()
{
  while (true) {
    get(C);
    get(F);
    get(D);
    // critical region:
    // use C, F, D
    release(C);
    release(F);
    release(D);
  }
}
```

4. Suppose the following two processes, foo and bar, are executed concurrently and share the semaphore variables S and R (each initialized to 1) and the integer variable x (initialized to 0).

```
void foo( ) {
    do {
      semWait(S);
      semWait(R);
      x++;
      semSignal(S);
      SemSignal(R);
    } while (1);
}
```

```
void bar( ) {
do {
    semWait(R);
    semWait(S);
    x--;
    semSignal(S;
    SemSignal(R);
} while (1);
}
```

a) Can the concurrent execution of these two processes result in one or both being blocked forever? If yes, give an execution sequence in which one or both are blocked forever.

b) Can the concurrent execution of these two processes result in the indefinite postponement of one of them? If yes, give an execution sequence in which one is indefinitely postponed.

5. Given the following state of a system:
The system comprises of five processes and four resources.
P1-P5 denotes the set of processes.
R1-R4 denotes the set of resources.
Total Existing Resources:

Existing			
R1	**R2**	**R3**	**R4**
6	3	4	3

	Allocation					**Claim**			
	R1	**R2**	**R3**	**R4**		**R1**	**R2**	**R3**	**R4**
P1	3	0	1	1		6	2	1	1
P2	0	1	0	0		0	2	1	2
P3	1	1	1	0		3	2	1	0
P4	1	1	0	1		1	1	1	1
P5	0	0	0	0		2	1	1	1

a) Compute the Available vector.
b) Calculate the Need matrix.
c) Is the current allocation state safe? If so, give a safe sequence of processes. In addition, show how the Available (working array) changes as each process terminates.
d) If the request (1,1,0,0) from Process P1 arrives, will it be correct to grant the request? Justify your decision.

Supplementary problems:

S1. Consider the following fragment of code on a Linux system.

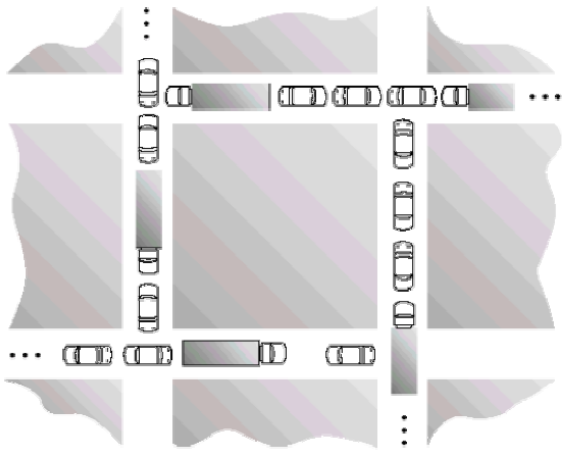
```
read_lock(&mr_rwlock);
write_lock(&mr_rwlock);
```

Where `mr_rwlock` is a *reader–writer* lock. What is the effect of this code?

S2. A computer science student assigned to work on deadlocks thinks of the following brilliant way to eliminate deadlocks. When a process requests a resource, it specifies a time limit. If the process blocks because the resource is not available, a timer is started.
If the time limit is exceeded, the process is released and allowed to run again. If you were the professor, what grade would you give this proposal and why?

S3. Is it possible to have a deadlock involving only one single-threaded process? Explain your answer.

S4. Consider the traffic deadlock depicted in the following figure.



a) Show that the four necessary conditions for deadlock indeed hold in this example.
b) State a simple rule that will avoid deadlocks in this system.