CvP - Werkcollege 13

Exercise 1 Consider an application that consists of the following four processes, where `...` represents some sequential computation, and `bexp1`, `bexp2`, and `bexp3` represent Boolean expressions. The `lock()` and `unlock()` calls, respectively, lock and unlock their binary semaphore arguments. The variables `S1`, `S2`, and `S3` are globally declared, shared binary semaphores.

```
Process A:
  ...
  lock(S1)
  lock(S2)
  lock(S3)
  ...
  unlock(S1)
  unlock(S2)
  unlock(S3)
  ...

Process B:
  ...
  lock(S3)
  if (bexp1) then {
      lock(S1)
      ...
      unlock(S1)
  }
  unlock(S3)
  ...

Process C:
  ...
  lock(S2)
  if (bexp2) then {
      ...
      lock(S3)
      ...
      if (bexp3) then {
          ...
```

```
            lock(S1)
            ...
            unlock(S1)
            ...
        }
        ...
        unlock(S3)
        ...
    }
    unlock(S2)
    ...
```

(a) For each process $X \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ and lock $S \in \{\mathrm{S1}, \mathrm{S2}, \mathrm{S3}\}$, draw the request/allocation graph where process X requests lock S and all other processes are in their initial state.

(b) Complete the following table:

A	B	C	acyclic?
S1	S3	S2	yes
S1	S3	S3	yes
	⋮		
S3	S1	S1	no

Each row corresponds to a possible configuration of the application. For example, the first row corresponds to the configuration where process A request lock S1, process B request lock S3, and process C requests lock S2. The last column states whether the associated RAG is acyclic.

(c) Find every possible deadlock in this application, and draw the associated request/allocation graph.

(d) Modify the code of these processes such that the application runs without any problem, while it still runs with the maximal degree of concurrency.

Exercise 2 Consider a concurrent application with $n \geq 2$ processes $P_1, \ldots, P_n$, and $k \geq 2$ resources $R_1, \ldots, R_k$. Each process requires a subset of resources during their execution. We say that P *request its resources in order* iff the following property holds: if P requires resources R_i and R_j, with $1 \leq i < j \leq k$, then P requests R_i before R_j.

Show that resource acquisition does not introduce a deadlock, if all processes request their resources in order.