1. Consider the following serializer code:

```c
serializer {
    queue a, b, c;
    crowd cr;
    int i = 0; /* initially zero */

    void foo() {
        enqueue(a) until i != 0;
        join_crowd(cr) {
            ... /* critical section */
        }
    }

    void bar() {
        enqueue(b) until a.empty() && i == 0;
        join_crowd(cr) {
            ... /* critical section */
        }
        i = 1;
    }

    void baz() {
        enqueue(c) until i == 1;
        join_crowd(cr) {
            ... /* critical section */
        }
        i = 0;
    }
}
```  // end of serializer

Which of the following path expression(s) describe(s) only executions for which the above serializer does not deadlock?

- a) path { foo + bar + baz } + bar end
- b) path { bar ; foo ; baz } end
- c) path bar ; { foo } ; baz end
- d) path { baz + foo } + bar end
e) path { bar ; baz } ; foo end

2. What happens if a Solaris user-level thread, as described in Stein and Shah, needs to be preempted while executing a blocking system call? What about a user-level thread running on top of the Psyche kernel?

3. Explain the performance of the ‘backoff ref’ spinlock algorithm observed in Figure 3 in Anderson’s Spinlock paper. Make sure to explain why this algorithm performs better/worse compared to other spinlock implementations at low/high loads.