(6 p) The following is a proposal for a solution to synchronize an arbitrary number of threads who need to access a shared resource, allowing at most C threads to access the resource at the same time. Argue about its safety and progress properties.

```
shared var
    S: binary semaphore init 1;
    doorway: binary semaphore init 1;
    count: integer init C;

thread-i {
    repeat

        [Other code]

        wait(doorway)
        wait(S)
        count := count - 1
        if count > 0 signal(doorway) end-if
        signal(S)

        [Code accessing the Shared Resource]

        wait(S)
        count := count + 1
        if count == 1 signal(doorway) end-if
        signal(S)

    forever

}
```

HINT: In the following, we use the terms "entry section" and "exit section" to refer to the respective parts of the code before and after the "Code accessing the Shared Resource".
Safety (no more than C threads can access the resource). Key arguments:
- The binary semaphore S ensures that the shared variable count updates are executed as a critical section. Once a thread A has passed the wait(S) statement, in either the entry or the exit section, A will: decrement or increment the count variable, conditionally execute a signal(doorway) operation, and then finally release the critical section that is ensuring that only one process can be updating the value in count.
- Through the conditions upon which the signal(doorway) operations are executed, it is ensured that the value of count is not allowed to become smaller than zero.
- This implies that if there are more than C threads trying to get access to the resource, the "excess" one(s) will become blocked on the wait(doorway) statement that is the first instruction in the entry section.
- This implies that only threads that can execute the entry section completely, i.e. when the count variable allows (via appropriate checks in the entry and the exit sections), are allowed to continue past the wait(doorway) statement.
Progress (ie no deadlock):
Following the above arguments and their implication that: if there exit(s) thread(s) waiting in the wait(doorway) trying to get access to the resources, once

the count variable allows, it/they will be allowed to continue past the wait(doorway) statement one by one, through corresponding signal(doorway) invocations in the exit section. i.e. there are no cyclical waits among threads.