

12 Concurrency

12.7 Exercises

- 12.33 In Section 12.4.1 we cast monitors as a mechanism for synchronizing access to shared memory, and we described their implementation in terms of semaphores. It is also possible to think of a monitor as a module inhabited by a single process, which accepts request messages from other processes, performs appropriate operations, and replies. Give the details of a monitor implementation consistent with this conceptual model. Be sure to include condition variables. (Hint: See the discussion of early reply in Section 12.2.3, page 597.)
- 12.34 Show how shared memory can be used to implement message passing. Specifically, choose a set of message-passing operations (e.g., no-wait send and explicit message receipt) and show how to implement them in your favorite shared-memory notation.
- 12.35 When implementing reliable messages on top of unreliable messages, a sender can wait for an acknowledgment message, and retransmit if it doesn't receive it within a bounded period of time. But how does the receiver know that its acknowledgment has been received? Why doesn't the sender have to acknowledge the acknowledgment (and the receiver acknowledge the acknowledgment of the acknowledgment...)? (For more information on the design of fast, reliable protocols, you might want to consult a text on computer networks [Tan02, PD07].)
- 12.36 An arm of an Occam ALT statement may include an *input guard*—a receive (?) operation—in which case the arm can be chosen only if a potential partner is trying to send a matching message. One could imagine allowing *output guards* as well: send (!) operations that would allow their arm to be chosen only if a potential partner were trying to receive a matching message. Neither Occam nor CSP (as originally defined) permits output guards. Can you guess why? Suppose you wished to provide them. How would the implementation work? (Hint: For ideas, see the articles of

Bernstein [Ber80], Buckley and Silbershatz [BS83b], Bagrodia [Bag86], or Ramesh [Ram87].)

- 12.37** In Section ©12.5.3 we described the semantics of a `terminate` arm on an Ada `select` statement: this arm may be selected if and only if all potential communication partners have terminated, or are likewise stuck in `select` statements with `terminate` arms. Erlang, Occam, and SR have no similar facility, though the original CSP proposal does. How would you implement `terminate` arms in Ada? Why do you suppose they were left out of Erlang, Occam, and SR? (Hint: For ideas, see the work of Apt and Francez [Fra80, AF84].)