5.20 In the sleeping barber problem (Section 5.2), suppose there are several barbers rather than just one. Develop a monitor to synchronize the actions of the customers and barbers. First specify a monitor invariant. The monitor should have the same procedures as in Figure 5.10. Be careful to ensure that `finished_cut` awakens the same customer that a barber rendezvoused with in `get_next_customer`. Use the Signal and Continue discipline.

5.15 Suppose `n` processes `P[1:n]` share two printers. Before using a printer, `P[i]` calls `request(printer)`. This operation returns the identity of a free printer. After using that printer, `P[i]` returns it by calling `release(printer)`.

(a) Develop a monitor that implements `request` and `release`. First specify a monitor invariant. Use the Signal and Continue discipline.

(b) Assume each process has a priority that it passes to the monitor as an additional argument to `request`. Modify `request` and `release` so that a printer is allocated to the highest priority waiting process. If two processes have the same priority, their requests should be granted in FCFS order.

5.10 *Atomic Broadcast.* Assume one producer process and `n` consumer processes share a bounded buffer having `b` slots. The producer deposits messages in the buffer; consumers fetch them. Every message deposited by the producer is to be received by all `n` consumers. Furthermore, each consumer is to receive the messages in the order they were deposited. However, consumers can receive messages at different times. For example, one consumer could receive up to `b` more messages than another if the second consumer is slow.

Develop a monitor that implements this kind of communication. Use the Signal and Continue discipline.

monitor invariant. Assume the arguments to `deposit` and `withdraw` are positive. Use the Signal and Continue discipline.

(b) Modify your answer to (a) so that withdrawals are serviced FCFS. For example, suppose the current balance is $200, and one customer is waiting to withdraw $300. If another customer arrives, he must wait, even if he wants to withdraw at most $200. Assume there is a magic function `amount(cv)` that returns the value of the amount parameter of the first process delayed on `cv`.

(c) Suppose a magic `amount` function does not exist. Modify your answer to (b) to simulate it in your solution.

5.8 *The Savings Account Problem.* A savings account is shared by several people (processes). Each person may deposit or withdraw funds from the account. The current balance in the account is the sum of all deposits to date minus the sum of all withdrawals to date. The balance must never become negative. A deposit never has to delay (except for mutual exclusion), but a withdrawal has to wait until there are sufficient funds.

(a) Develop a monitor to solve this problem. The monitor should have two procedures: `deposit(amount)` and `withdraw(amount)`. First specify a

5.6 Consider the dining philosophers problem defined in Section 4.3.

(a) Develop a monitor to implement the required synchronization. The monitor should have two operations: `getforks(id)` and `relforks(id)`, where `id` is the identity of the calling philosopher. First specify a monitor invariant, then develop the body of the monitor. Your solution need not be fair. Use the Signal and Continue discipline.

(b) Modify your answer to (a) so that it is fair—i.e., so that a philosopher who wants to eat eventually gets to.

5.4 The following problems deal with the readers/writers monitor in Figure 5.5. Assume Signal and Continue semantics for all four problems.

(a) Suppose there is no `signal_all` primitive. Modify the solution so that it uses only `signal`.

(b) Modify the solution to give writers preference instead of readers.

(c) Modify the solution so that readers and writers alternate if both are trying to access the database.

(d) Modify the solution so that readers and writers are given permission to access the database in FCFS order. Allow readers concurrent access when that does not violate the FCFS order of granting permission.

5.3 Consider the following proposed solution to the shortest-job-next allocation problem in Section 5.2:

```
monitor SJN {
  bool free = true;
  cond turn;
  procedure request(int time) {
    if (!free)
      wait(turn, time);
    free = false;
  }
  procedure release() {
    free = true;
    signal(turn);
  }
}
```

Does this solution work correctly for the Signal and Continue discipline? Does it work correctly for Signal and Wait? Clearly explain your answers.