#1: p. 237-238, Problem 7.9:

Semaphore tobacco = 0, 
paper = 0, 
matches = 0, 
agent = 0;

agent() {
    while(1) {
        choose2(paper, tobacco, matches);
        if(!paper) {
            signal(paper);
        }
        else if(!matches) {
            signal(matches);
        }
        else if(!tobacco) {
            signal(tobacco);
        }
        wait(agent)
    }
}

paper() {
    while(1) {
        wait(paper);
        pickup2(matches, tobacco);
        signal(agent);
        smoke();
    }
}

tobacco() {
    while(1) {
        wait(tobacco);
        pickup2(paper, matches);
        signal(agent);
        smoke();
    }
}
matches() {
    while(1) {
        wait(matches);
        pickup2(paper,tobacco);
        signal(agent);
        smoke();
    }
}

#2: p. 239, Problem 7.11:

monitor embedded {
    semaphore full, empty;
    queue Q;

    void push(process) {
        while(Q.isFull())
            wait();
        Q.push(process);
        Q.signal();
    }

    process pop(process) {
        while(Q.isEmpty())
            wait();
        Q.pop(process);
        Q.signal();
        return process;
    }
}

#3: p. 239, Problem 7.12a:

Since the monitor in the previous question enforces strict mutual exclusion, it only allows one process to access the data at a time. This is good for single processor systems, but this would severely hinder the performance of multi-processor systems that split the execution of a process between processors. During this type of execution, only one processor running the process would be allowed to run at any given time, the other processors would have to wait. Also, if the amount of processes trying to run in the system is large, the wait time would be increased.
monitor AlarmClock {
    int now = 0;
    semaphor TimeToWakeup;

    void SetAlarm (int delay) {
        for(int i=0; i < delay; i++)
            tick();
    }

    void tick() {
        now++;
        TimeToWakeup.signal;
    }
}