If a system crash occurs, the information in the log is used in restoring the state of the updated data items, which is accomplished with the use of the undo and redo operations. To reduce the overhead in searching the log after a system failure has occurred, we can use a checkpoint scheme.

When several transactions overlap their execution, the resulting execution may no longer be equivalent to an execution where these transactions executed atomically. To ensure correct execution, we must use a concurrency-control scheme to guarantee serializability. There are various different concurrency-control schemes that ensure serializability by either delaying an operation or aborting the transaction that issued the operation. The most common ones are locking protocols and timestamp-ordering schemes.

### Exercises

7.1 What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

7.2 Explain why spinlocks are not appropriate for uniprocessor systems yet may be suitable for multiprocessor systems.

7.3 Prove that, in the bakery algorithm (Section 7.2.2), the following property holds: If $P_i$ is in its critical section and $P_k$ ($k \neq i$) has already chosen its number $[k] \neq 0$, then $(\text{number}[i], 1) < (\text{number}[k], k)$.

7.4 The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, $P_0$ and $P_1$, share the following variables:

```c
boolean flag[2]; /* initially false */
int turn;
```

The structure of process $P_i$ ($i == 0$ or $1$), with $P_j$ ($j == 1$ or $0$) being the other process, is shown in Figure 7.27.

Prove that the algorithm satisfies all three requirements for the critical-section problem.

7.5 The first known correct software solution to the critical-section problem for $n$ processes with a lower bound on waiting of $n - 1$ turns was presented by Eisenberg and McGuire. The processes share the following variables:

```c
enum pstate {idle, want.in, in.cs};
pstate flag[n];
int turn;
```

7.6 The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, $P_0$ and $P_1$, share the following variables:

```c
enum pstate {idle, want.in, in.cs};
pstate flag[n];
int turn;
```
do {
    flag[i] = true;
    while (flag[i]) {
        if (turn == j) {
            flag[i] = false;
            while (turn == j);
            flag[i] = true;
        }
    }
}

critical section

    turn = j;
    flag[i] = false;

remainder section
}

Figure 7.27 The structure of process $P_i$ in Dekker’s algorithm.

All the elements of flag are initially idle; the initial value of turn is immaterial (between 0 and n-1). The structure of process $P_i$ is shown in Figure 7.28.

Prove that the algorithm satisfies all three requirements for the critical-section problem.

7.6 In Section 7.3, we mentioned that disabling interrupts frequently can affect the system’s clock. Explain why it can, and how such effects can be minimized.

7.7 Show that, if the wait and signal operations are not executed atomically, then mutual exclusion may be violated.

7.8 The Sleeping-Barber Problem. A barbershop consists of a waiting room with $n$ chairs and the barber room containing the barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Write a program to coordinate the barber and the customers.

7.9 The Cigarette-Smokers Problem. Consider a system with three smoker processes and one agent process. Each smoker continuously rolls a cigarette and then smokes it. But to roll and smoke a cigarette, the smoker