Ch7 Practice Exercises

7.4 Describe how deadlock is possible with the dining-philosophers problem.

Answer:

1) The philosopher can only use the chopstick on his or her immediate left or right. The philosophers never speak to each other, which creates a dangerous possibility of deadlock.
2) Deadlock arises when all of the philosophers decide to eat at the same time. For example, every philosopher holds a left chopstick and waits perpetually for a right chopstick (or vice versa).

Ch7 Exercises

7.8 The Linux kernel has a policy that a process cannot hold a spinlock while attempting to acquire a semaphore. Explain why this policy is in place.

Answer:

1) Because acquiring a semaphore may put the process to sleep while it is waiting for the semaphore to become available. Spinlocks are to only be held for short duration, and a process that is sleeping may hold the spinlock for too long a period.
2) For example, if thread A sleeps in a spinlock, and thread B then tries to acquire the same spinlock, a uniprocessor system will deadlock. Thread B will never go to sleep (because spinlocks don't have the waitlist necessary to awaken B when A is done), and thread A will never get a chance to wake up.

Ch7 Programming

7.9 Design an algorithm for a bounded-buffer monitor in which the buffers (portions) are embedded within the monitor itself.

Answer:

Algorithm:

```
monitor boundedbuffer {
   const bufsize = 100;
   item buf[100];
   int in, out, count;
   condition Producer, Consumer;

   void produce (item x) {
       if (count == bufsize) Producer.wait();
       buf[in] = x;
       count++;
       in = (in + 1) % n;
       Consumer.signal();
   }
   void consume (item& x) {
       if (count == 0) Consumer.wait();
       x = buf[out];
       count --;
       out = (out + 1) % n;
       Producer.signal();
   }
   initialization_code() {
       in:=0;
       out:=0;
       count:=0;
   }
}
```

Explanation:

1) While the count doesn't meet the limitation, the function "produce()" could always work, and could always send signal to "consume()". So do the function "consume()". Thus it makes an mutual exclusion within a monitor, namely, the buffers are embedded within the monitor itself.
2) However, this bounded-buffer monitor is mainly suitable for small portions.
3) For example, if the portions are large, then adding and removing data to the shared pool requires a considerable amount of time. Since the producer and consumer cannot execute in the monitor concurrently, parallelism is lost.

Ch8 Exercises

8.23 Consider a system consisting of m resources of the same type being shared by n threads. A thread can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each thread is between one resource and m resources.

b. The sum of all maximum needs is less than $m + n$.

Answer:

Conditions:

a. $1 \leq Max_i \leq m$ for all i

b. $\sum_{i=1}^{n} Max_i < m + n$

Need to prove: The system is deadlock free.

Proof: I will prove by contradiction.

1) Assume there is a deadlock, means that $\sum_{i=1}^{n} Allocation_i = m$

2) Since we know: $Need_i = Max_i - Allocation_i$

3) Combing <b>, we get: $\sum_{i=1}^{n} Need_i + \sum_{i=1}^{n} Allocation_i = \sum_{i=1}^{n} Max_i < m + n$

4) Combing <1>, we get: $\sum_{i=1}^{n} Need_i < n$

5) This implies that there exists a process P_i such that $Need_i = 0$

6) Following <a>, we get P_i has at least one resource that it can release.

7) Thus there is a contradiction between <5> and <6>.

8) Proof is done, the system cannot be in a deadlock state.

8.24 Consider the version of the dining-philosophers problem in which the chopsticks are placed at the center of the table and any two of them can be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request can be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

Answer:

When a philosopher makes a request for the first chopstick, do not satisfy the request only if there are no other philosopher with two chopsticks and if there is only one chopstick remaining.