PREV
B. Review of Mathematica

NEXT
D. Software Tools

Appendix C. Concurrent Programming Problems

This appendix contains a list of problems to be solved by writing concurrent programs. For completeness, the list includes problems presented in the text, together with a reference to the sections where they were first posed.

1. The critical section problem (Section 3.2).
2. The producer–consumer problem (Section 6.7).
3. The problem of the readers and writers (Section 7.6). When solving the problem, you must specify if you want to give precedence to readers or to writers, or to alternate precedence to ensure freedom from starvation.
4. Multiple readers and writers. In the problem of the readers and writers, the two classes of processes were distinguished by allowing multiple readers but not multiple writers. We now allow multiple writers. Develop an algorithm to coordinate reading and writing, paying special attention to freedom from starvation. (This problem is also known as the unisex bathroom problem and the baboon bridge-crossing problem.)
5. Conway's problem (Section 8.2).
6. The sleeping barber. Consider the following scenario for Algorithm 6.7 for the producer–consumer problem:

n	producer	consumer	Buffer

2	**signal(notEmpty)**	wait(notEmpty)	[1]
3	append(d, Buffer)	**wait(notEmpty)**	[1]
4	append(d, Buffer)	**d ← take(Buffer)**	[1]
5	**append(d, Buffer)**	wait(notEmpty)	[ ]

Line 5 is the same as line 1 and the scenario can be extended indefinitely.

Every time that the `consumer` process executes `wait`, the value of the semaphore is nonzero so it never blocks. Nevertheless, both processes must execute semaphore operations, which are less efficient than ordinary statements. Develop an algorithm to solve the producer–consumer problem so that the consumer executes `wait` only if it actually has to wait for the buffer to be nonempty.

7. (Patil, Parnas) The cigarette smokers problem. The problem concerns three resources, three servers and three clients. Server `S12` can supply resources r_1 and r_2, server `S13` can supply resources r_1 and r_3, and server `S23` can supply resources r_2 and r_3. Similarly, client `C12` needs resources r_1 and r_2, client `C13` needs resources r_1 and r_3, and client `C23` needs resources r_2 and r_3.[1] The servers supply resources one at a time and notify accordingly, so clients cannot simply wait for pairs of resources to become available. For example, server `S23` may execute `signal(r2)` to indicate that resource r_2 is available, but this may unblock client `C12` waiting for its second semaphore or client `C23` waiting for its first semaphore. The original statement of the problem asks for a solution using only semaphore operations without conditional statements.

8. (Manna and Pnueli) Computation of the binomial coefficient:

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!} = \frac{n \cdot (n-1) \cdot \cdots \cdot (n-k+1)}{1 \cdot 2 \cdot \cdots \cdot k}.$$

Let one process compute the numerator and the other the denominator. Prove that $i!$ divides $j \cdot (j+1) \cdot \cdots \cdot (j+i-1)$. Therefore, the numerator process can receive partial results from the denominator process and do the division immediately, keeping the intermediate results from becoming too big. For example, $1 \cdot 2$ divides $10 \cdot 9$, $1 \cdot 2 \cdot 3$ divides $10 \cdot 9 \cdot 8$ and so on.

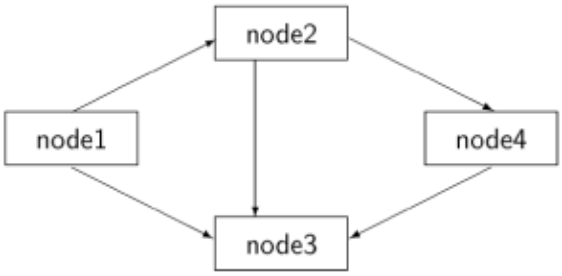
9. The dining philosophers (Section 6.9).

10. The roller-coaster problem. There are n people at an amusement park who can decide they want to ride the roller-

coaster, or they can be engaged in other activities and may never try to ride the roller-coaster. A single-car roller-coaster ride begins its run only when exactly *r* passengers are ready. Develop an algorithm to synchronize the actions of the roller-coaster and the passengers. Generalize the program to multiple roller-coaster cars; however, the cars cannot pass each other.

11. (Trono) The Santa Claus problem. Develop an algorithm to simulate the following system: Santa Claus sleeps at the North Pole until awakened by either all of the nine reindeer, or by a group of three out of ten elves. He then performs one of two indivisible actions: If awakened by the group of reindeer, Santa harnesses them to a sleigh, delivers toys, and finally unharnesses the reindeer who then go on vacation. If awakened by a group of elves, Santa shows them into his office, consults with them on toy R&D, and finally shows them out so they can return to work constructing toys. A waiting group of reindeer must be served by Santa before a waiting group of elves. Since Santa's time is extremely valuable, marshaling the reindeer or elves into a group must not be done by Santa.

12. Consider an arbitrary directed graph with no cycles and at least one source node, for example:



The graph is intended to represent precedence requirements: the computation at a node cannot begin until the computations at previous nodes have been completed. Develop an algorithm that takes a representation of a precedence graph as an input and generates a synchronization skeleton that ensures that the precedence requirements are respected.

13. Barriers.[2] In Section 6.6 we showed how to use semaphores to solve order-of-execution problems and the previous problem asked you to generalize this to arbitrary precedence relations. The *barrier* problem is another generalization, in which a group of processes have to repeatedly synchronize phases of their computation (Algorithm C.1). Barrier synchronization is important in parallel scientific computation on multiprocessors. Solve the problem efficiently using the constructs in Section 3.10.

14. Sorting algorithms. Section 6.6 briefly mentioned mergesort, but many other sorting algorithms can be written to run concurrently, in particular those like quicksort that work by divide and conquer.

Table C.1. Barrier synchronization

```
global variables for synchronization
```

```
      loop forever
p1:     wait to be released
p2:     computation
p3:     wait for all process to finish their computation
```

15. (Hoare) Develop an algorithm for a server that minimizes the amount of seek time done by the arm of a disk drive. A simple server could satisfy requests for data in decreasing order of distance from the current position of the arm. Unfortunately, this could starve a request for data if requests for close tracks arrive too fast. Instead, maintain two queues of requests: one for requests for data from track numbers less than the current position and one for requests with higher track numbers. Satisfy all requests from one queue before considering requests from the other queue. Make sure that a stream of requests for data from the current track cannot cause starvation.

16. Matrix multiplication (Sections 8.3 and 9.4).

17. (Roussel) Given two binary trees with labeled leaves, check if the sequence of labels is the same in each tree. For example, the two trees defined by the expressions (*a,* (*b, c*)) and ((*a, b*), *c*) have the same sequence of leaves. Two processes will traverse the trees concurrently, sending the leaf labels in the order encountered to a third process for comparison.

18. (Dijkstra) Let S and T be two disjoint sets of numbers. Develop an algorithm that modifies the two sets so that S contains the $|S|$ smallest members of $S \cup T$ and T contains the $|T|$ largest members of $S \cup T$. One process will find the largest element in S and send it to a second process that returns the smallest element in T.

19. (Conway) The Game of Life. A set of cells is arranged in a (potentially infinite) rectangular array so that each cell has eight neighbors (horizontally, vertically and diagonally). Each cell is *alive* or *dead*. Given an initial finite set of alive cells, compute the configuration obtained after a sequence of generations, using a separate process for each cell or group of cells. The rules for passing from one generation to the next are:

    1. If a cell is alive and has fewer than two live neighbors, it dies.
    2. If it has two or three live neighbors, it continues to live.
    3. If it has four or more live neighbors, it dies.
    4. A dead cell with exactly three live neighbors becomes alive.

20. Given a binary tree with nodes labeled by numbers, compute the sum of the labels in the tree.

21. Given a binary tree representing the parse tree of an arithmetical or logical expression, evaluate the expression.

22. Computation of all prime numbers from 2 to n. First prove that if k is not a prime, it is divisible by a prime $p(k) \leq \lfloor \sqrt{k} \rfloor$. The algorithm called the Sieve of Eratosthenes can be written as a concurrent program using dynamic creation of processes. Allocate a process to delete all multiples of 2. Whenever a number is discovered to be prime by all existing processes, allocate a new process to delete all multiples of this prime.

23. Image processing is a natural application area for concurrent programming. Typically, discrete integral and differential operators are applied independently at each pixel in the image. For example, to smooth an image, replace the pixel at $x_{i,j}$ with the weighted average of the pixel and its four neighbors:

    $(4 \cdot x_{i,j} + x_{i+1,j} + x_{i,j+1} + x_{i-1,j} + x_{i,j-1})/8$.

    To sharpen an image, replace the pixel with:

    $(4 \cdot x_{i,j} - x_{i+1,j} - x_{i,j+1} - x_{i-1,j} - x_{i,j-1})/8$.

24. Solutions to the critical section problem enable processes to obtain exclusion access to a single resource. In the *resource allocation problem*, there is a set of n (identical) resources that m processes can request, use and then return. (Of course the problem is meaningful only if $m > n$; otherwise, a resource could be allocated permanently to each process.) A server allocates available resources and blocks clients when no more resources are available. Solve the resource allocation problem, first for the case where each process requests only a single resource at a time, and then for the case where a process can request $k \leq n$ resources at a time. The latter problem is difficult to solve because of the need to ensure atomicity of execution from the request to the allocation of the resources, and furthermore, if the process is blocked due to lack of resources, the request must be retried or resubmitted when resources become available.

25. The *stable marriage problem* concerns matching two sets of items, in a manner that is consistent with the preferences of each item [30]. A real-world application is matching between hospitals offering residencies and medical students applying to the hospitals. The problem itself is conventionally expressed in terms of matching men and women:

    Given a set of n men and n women, each man lists the women in order of preference and each woman lists the men in order of preference. A *matching* is a list of n pairs (m, w), where the first element is from the set of men and second from the set of women. A matching is *unstable* if there exists a pair of matches (m_1, w_1) and (m_2, w_2) such that m_1 prefers w_2 to w_1 and also w_2 prefers m_1 to m_2. A matching is *stable* if it is not unstable. Find an algorithm to construct a stable matching.

    Here is an example of preference lists, where decreasing preference is from left to right in each line:

Man	List of women

1	2	4	1	3
2	3	1	4	2
3	2	3	1	4
4	4	1	3	2

Wo-man	List of men			
1	2	1	4	3
2	4	3	1	2
3	1	4	3	2
4	2	1	4	3

Any matching containing the pairs (1, 1) and (4, 4) is unstable, because man 1 prefers woman 4 to woman 1 and woman 4 prefers man 1 to man 4. There are two stable matchings for this instance of the problem:

{(1, 4), (2, 3), (3, 2), (4, 1)}, {(1, 4), (2, 1), (3, 2), (4, 3)}.

Table C.2. Gale–Shapley algorithm for a stable marriage

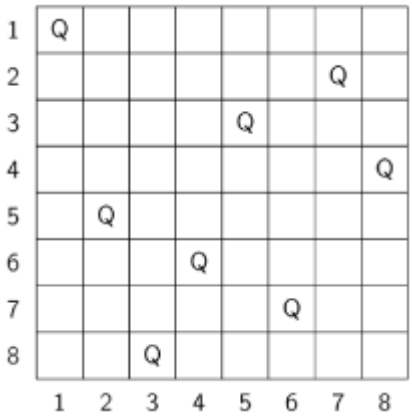
```
integer list freeMen ← {1,.. . ,n}

integer list freeWomen ← {1,.. . ,n}

integer pair-list matched ← ∅

integer array[1..n, 1..n] menPrefs ←. ..

integer array[1..n, 1..n] womenPrefs ←. ..

integer array[1..n] next ← 1
```

```
p1: while freeMen ≠ ∅, choose some m from freeMen
p2:   w ← menPrefs[m, next[m]]
p3:   next[m] ← next[m] + 1
p4:   if w in freeWomen
p5:      add (m,w) to matched, and remove w from freeWomen
p6:   else if w prefers m to m' // where (m',w) in matched
p7:      replace (m',w) in matched by (m,w), and remove m' from freeMen
p8:   else // w rejects m, and nothing is changed
```

Algorithm C.2 is the *Gale–Shapley* algorithm for constructing a stable matching. (The implementation of `w prefers m to m'` is not given; it is easy to program using the preference matrices or by constructing auxiliary arrays that directly return the answer to this question.) The algorithm has the remarkable property that for any instance of the problem, it will find a stable matching, and the same matching will be found regardless of the order in which the men are chosen.

Develop a concurrent algorithm to solve the stable marriage problem. Is your algorithm significantly more efficient than a sequential one?

26. The *n-Queens problem* is to place n queens on an $n \times n$ board so that no queen can take another according to the rules of chess. Here is an example of a solution of the 8-queens problem:



Develop a concurrent algorithm to find one solution (or all solutions) to the n-queens problem. There are several different ways of utilizing the inherent concurrency in the problem. A separate process can search for a solution starting from each of the n possible assignments of a queen to the first column. Or, each time a queen is successfully placed on the board, a new process can be allocated to continue expanding this potential solution.

27. Develop a distributed algorithm for leader election in a ring of n nodes. In a ring, each process can communicate only with its successor and predecessor nodes (or, in another version of the problem, only with its successor node). Eventually, exactly one node outputs a notification that it is the leader.

[1] The problem was originally described in terms of smokers needing tobacco, paper and matches;

in the interests of public health, it is presented in terms of abstract resources.

[2] There is no connection between this problem and the concept of barriers of protected objects (Section 7.10).

PREV
B. Review of Mathematical Logic

NEXT
D. Software Tools