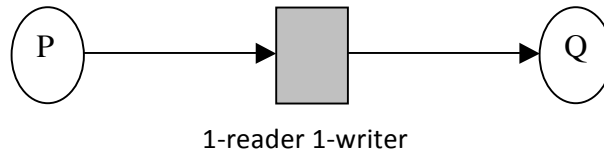


Solution to Chapter 3 Exercises

3.1. In the link-register model, it is possible for the sender process P to modify the data after writing into it once, if the receiver process Q has not read that value yet. Using the *message-passing* model with a unit-capacity channel and blocking send, the sender cannot modify the data that has been sent. The sender is unblocked when the receiver received it. With non-blocking send, the result will depend on the buffer management policy of the receiver.



3.2. *Keyboard K*

repeat

Send character(s) to the channel (K,P)

until blocked

Process P

repeat

Execute a step of the computation;

if channel (K,P) is not empty **then**

receive character(s) & store them in memory

else skip

forever

- (a) With non-blocking receive, the computation progresses without any problem.
- (b) With blocking receive, the receiver has to wait until the channel becomes non-empty (or the last character has been sent by K), before resuming the steps of the computation. Progress is slower, and the steady state progress is determined by the speed at which K sends data.

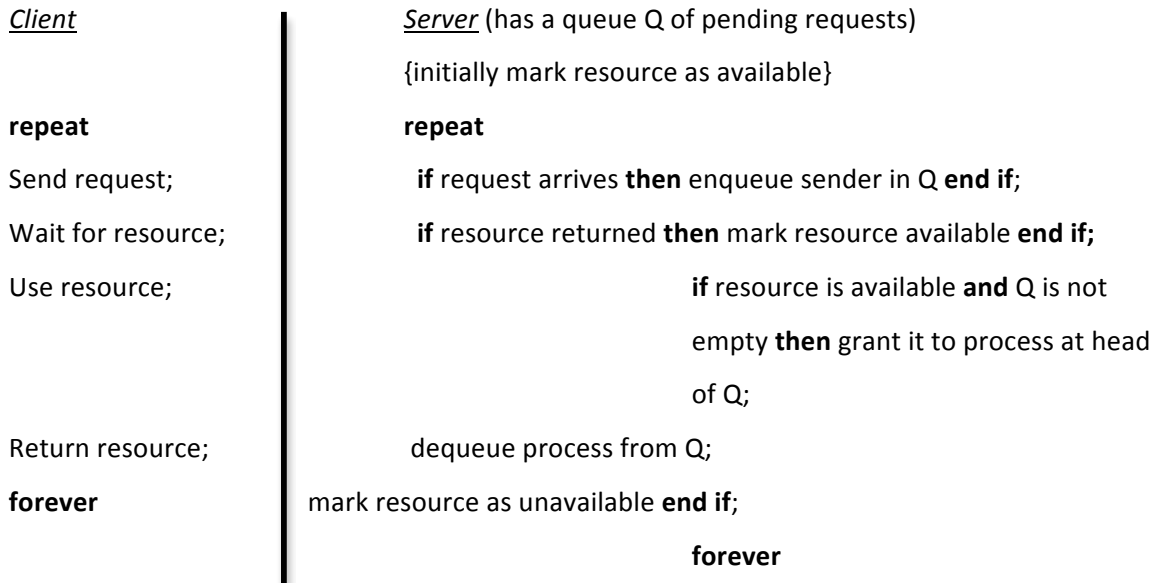
3.3. P maintains the array $a[0..\text{max}-1]$ and Q maintains the array $y[0..\text{max}-1]$

P:: **repeat** move! $a[k]$; $k := k+1$ **until** $k = \text{max}$

move:: **repeat** P? x; Q! x **forever**

Q:: **repeat** move? x; $y[k] := x$; $k := k+1$ **forever**

3.4. Sample client-server communication using *message-passing* model: the number of clients $N > 1$. Assume that there is a single resource managed by the server. Let fair scheduling conform to the order of the requests received by the server.



3.5. Assume that the maximum temperature is not the instantaneous maximum, but the maximum value recoded over a period of time. Let *maxtemp* be the maximum temperature. Each sensor executes the following steps:
 {initially *maxtemp* := locally recorded temperature}
 {in each round} broadcasts the locally recorded temperature, collects the values recorded by other sensors, and sets *maxtemp* to {*maxtemp* and the largest of the recorded values}. If instead the instantaneous max is required, then *maxtemp* is set to the maximum of the recorded values at the end of each round of broadcast.

3.6. Let there be n processes, and each identifier consist of k bits. In each round, every process *flips a coin*, and broadcasts the outcome of the coin toss to all. If all tosses return identical values, then no progress is made. Otherwise processes that return a *head* set the most significant bit of their ids to 1 (call it group 1), those returning a *tail* set the most significant bit of their ids to 0 (call it group 0). These steps are iterated within each group to create subdivisions within the group, until each group contains at most one process, and the values of the remaining $(k-1)$ bits of the id are computed.

Within a group, the id of each process will have identical prefix. The algorithm will terminate when each group has a single member, so uniqueness of the id is guaranteed.

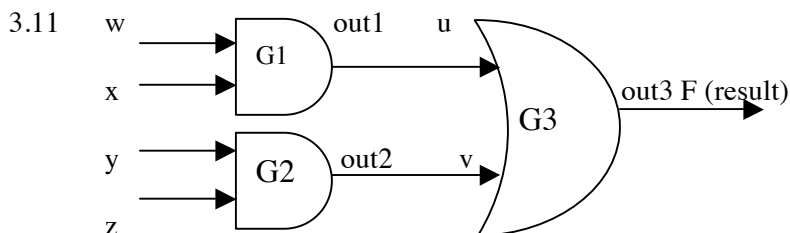
3.7. We use bounded delay channels, and assume that an upper bound of the diameter (if the network is connected) is known. P broadcasts the query “is Q there?” If Q receives it, then it responds by a broadcast “I am Q.” Every recipient (other than Q) forwards the query, and any process (other than P) that receives a response, broadcasts it to all neighbors *at most once*. If the channels have a bounded delay δ and there exists a path between P and Q, then in a bounded time ($2 \times \text{diameter} \times \delta$) P will receive the response from Q. The positive response will propagate back to P using the same mechanism as the query.

3.8. This is *knowledge-based* communication. Communication is possible only because both Alice and Bob share common knowledge (here it is the interpretation of the absence of a message) and make use of it.

3.9. This is also knowledge-based communication – the secret is never directly transmitted. However it relies on perfectly synchronized clocks. With approximately synchronized clocks, it will not work. The recipient can learn the approximate value of the secret (as an element of a finite set of values), but it may be useless.

*3.10. There are several published algorithms, for example see

Ahmadi, M. Stone, P. A Distributed Biconnectivity Check. *8th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, Minneapolis, Minnesota, USA, July 2006.



(i) Representation of the above circuit using CSP

G1:: *[out1:=AND (w, x); G3! out1]

G2:: *[out2:= AND (y, z); G3! out2]

G3:: *[G1? u; G2? v; out3 := OR(u, v); F! out3]

F:: G3? result

(Each gate, as well as the final output F is a process in the above representation.)

(ii) Treat each link as a 1-writer 1-reader register. The sending gate (or the input process) will write into this register and the receiving gate (or the output process) will read from this register.

3.12. {program for process $i > 0$ }

busy[i]: Boolean {indicates that process i is executing task[i]}

ack[i]: Boolean {indicates ack send to stage i-1}

task[i]: task to be performed by process i

done[i]: Boolean {indicated that process i has completed its task}

(initially, for all i: busy[i]=false, done[i] = false; ack[i]:=true)

if \neg busy[i] **and** done[i-1] **and** ack[i+1] **then**

 done[i]:=false; busy[i] := true;

 execute task [i];

 busy[i]:= false; done[i]:= true

 ack[i]:=true

end if

{the program for the first and the last stages will be slightly different}

3.14. Assume a completely connected topology. Let phase[i] be the phase of process i. Let done[i.k] be a Boolean that indicates that process [i] has completed phase k. Initially, $\forall i$, phase[i] = 0, and done[i,k] = false for all $k \geq 0$.

{Program for process i, phase k, initially k=0}

repeat

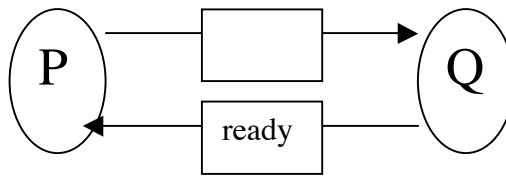
 start phase k;

```

        end phase k; done[i,k]:=true
        if  $\forall j \neq i: \text{done}[j,k]$ 
        then start phase[k+1]
        else skip;
    forever

```

3.15. Process Q indicates its readiness to receive the message by placing a ready message on the link to P



Process P

```

repeat
    while ready message does not arrive from Q do skip;
    send a message
forever

```

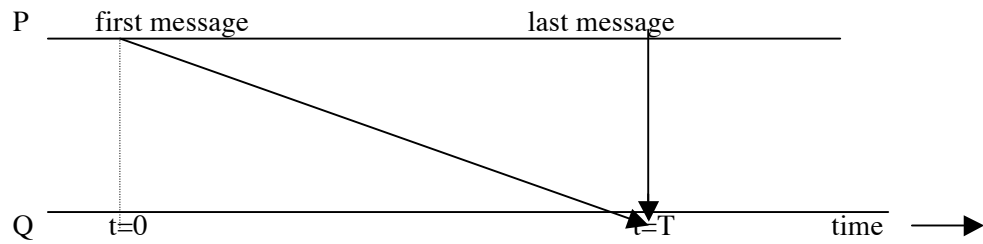
Process Q

```

send ready message;
repeat
    while a message does not arrive from P do skip
    receive the message; send ready message
forever

```

3.16. Buffering is needed when the messages do not reach the destination in the proper order. In the worst case, let the messages sent by P arrive at Q in the reverse temporal order. Thus assume that the last message reaches Q in zero time, but the first message takes T seconds. Before the arrival of the first message, all other messages that reach Q must be saved in a buffer. During this interval, $(r.T-1)$ messages must have been sent out. Thus the smallest buffer size has to be $(r.T-1)$ in the worst case.



3.17. The general answer is NO. Named systems use ids as a part of the algorithm. In a named system of size n , to store its own id or the id of a neighbor, a process will take at least $\log_2 n$ bits.

3.18. (Part 1) Assume that no two senders send to the same recipient. The following scenario holds:

Round 0: one person has the message,

Round 1: 2 persons have the message

Round 3: 4 persons have the message, and so on.

Thus, in $\lceil \log n \rceil$ rounds, the entire population of size n will receive the message.