```
type kind = enum(reqP, reqV, VOP, POP, ACK);
chan semop[n](int sender; kind k; int timestamp);
chan go[n](int timestamp);

process User[i = 0 to n-1] {
  int lc = 0, ts;
  ...
  # ask my helper to do V(s)
  send semop[i](i, reqV, lc); lc = lc+1;
  ...
  # ask my helper to do P(s), then wait for permission
  send semop[i](i, reqP, lc); lc = lc+1;
  receive go[i](ts); lc = max(lc, ts+1); lc = lc+1;
}

process Helper[i = 0 to n-1] {
  queue mq = new queue(int, kind, int);   # message queue
  int lc = 0, s = 0;         # logical clock and semaphore
  int sender, ts; kind k;    # values in received messages
  while (true) {   # loop invariant DSEM
    receive semop[i](sender, k, ts);
    lc = max(lc, ts+1); lc = lc+1;
    if (k == reqP)
      { broadcast semop(i, POP, lc); lc = lc+1; }
    else if (k == reqV)
      { broadcast semop(i, VOP, lc); lc = lc+1; }
    else if (k == POP or k == VOP) {
      insert (sender, k, ts) at appropriate place in mq;
      broadcast semop(i, ACK, lc); lc = lc+1;
    }
    else {   # k == ACK
      record that another ACK has been seen;
      for (all fully acknowledged VOP messages in mq)
        { remove the message from mq; s = s+1; }
      for (all fully acknowledged POP messages in mq st s > 0) {
        remove the message from mq; s = s-1;
        if (sender == i)    # my user's P request
          { send go[i](lc); lc = lc+1; }
      }
    }
  }
}
```

Figure 9.13 Distributed semaphores using a broadcast algorithm.