※ 序列协调原语

回顾

- sender-receiver的轮询（polling）
    - 能否提高效率?
    - 考虑一种条件唤醒机制，比如仅在in-out < N时唤醒
        - 节省了轮询所消耗的CPU时钟周期
    - 方法：线程序列协调原语(sequence coordination)

※ 序列协调原语

朴素实现

- 线程表中
  - 状态增加waiting，属性增加event
- 原语
  - wait (event_name)：
    - 线程状态=waiting，event设为event_name
  - notify (event_name)：
    - 将包含event_name的waiting状态的线程设为runnable

※ 序列协调原语

实现

```
shared structure buffer                              // A shared bounded buffer
    message instance message[N]                      // with a maximum of N messages
    long integer in initially 0                      // Counts number of messages put in the buffer
    long integer out initially 0                     // Counts number of messages taken out of the buffer
    lock instance buffer_lock initially UNLOCKED     // Lock to coordinate sender and receiver
    event instance room                              // Event variable to wait until there is room in buffer
    event instance notempty                          // Event variable to wait until the buffer is not empty

procedure SEND (buffer reference p, message instance msg)
    ACQUIRE (p.buffer_lock)
    while p.in − p.out = N do                        // Wait until there room in the buffer
        RELEASE (p.buffer_lock)                      // Release lock so that receiver can remove
        WAIT(p.room)                                 // Release processor
        ACQUIRE (p.buffer_lock)
    p.message[p.in modulo N] ← msg                   // Put message in the buffer
    if p.in = p.out then NOTIFY(p.notempty)          // Signal thread that there is a message
    p.in ← p.in + 1                                  // Increment in
    RELEASE (p.buffer_lock)

procedure RECEIVE (buffer reference p)
    ACQUIRE (p.buffer_lock)
    while p.in = p.out do                            // Wait until there is a message to receive
        RELEASE (p.buffer_lock)                      // Release lock so that sender can add
        WAIT(p.notempty)                             // Release processor
        ACQUIRE (p.buffer_lock)
    msg ← p.message[p.out modulo N]                  // Copy item out of buffer
    if p.in − p.out = N then NOTIFY(p.room)          // Signal thread that there is room now
    p.out ← p.out + 1                                // Increment out
    RELEASE (p.buffer_lock)
    return msg
```