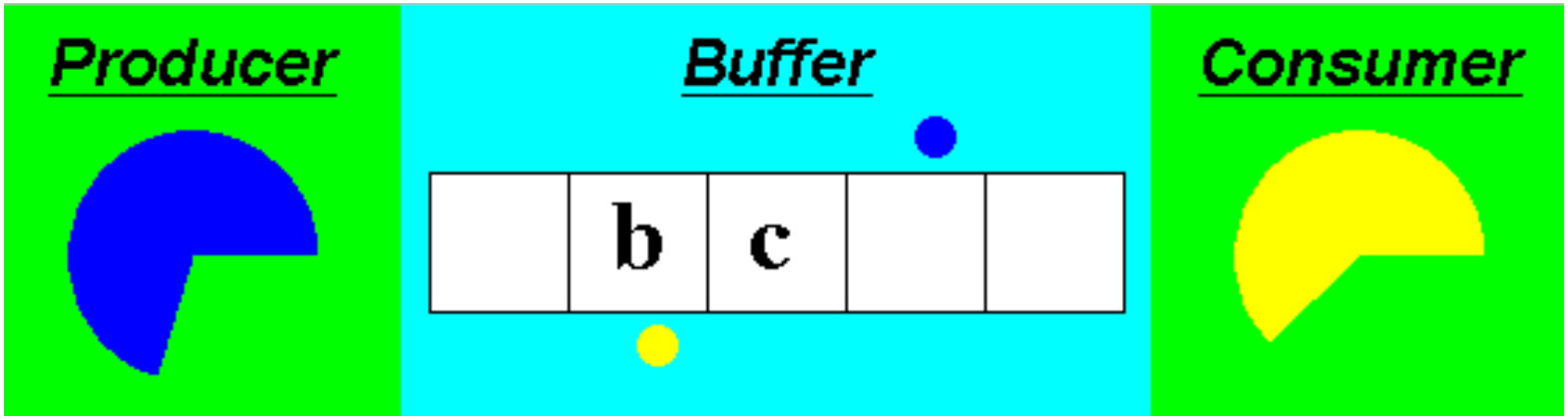
Semaphores, Monitors and Buffers

CS 2SD3

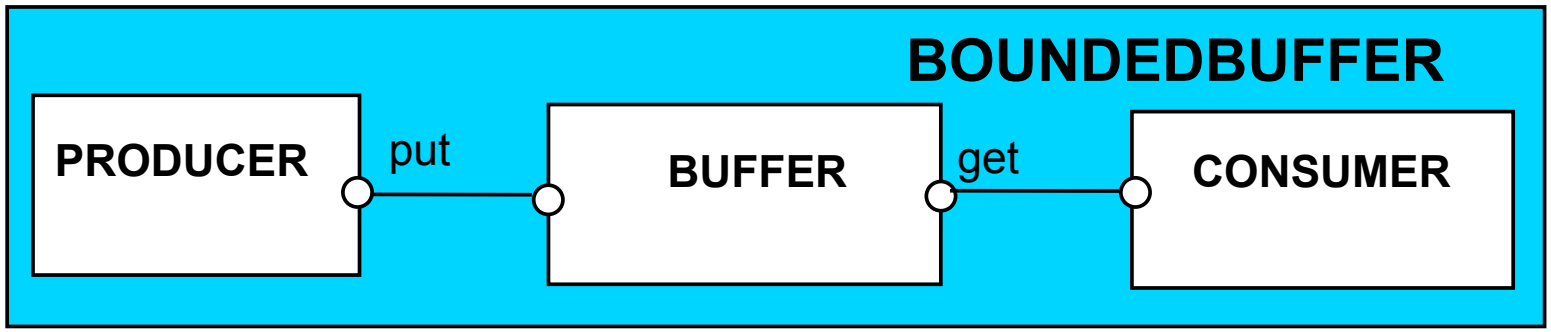
Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Bounded Buffer



- A bounded buffer consists of a fixed number of slots. Items are put into the buffer by a *producer* process and removed by a *consumer* process. It can be used to smooth out transfer rates between the *producer* and *consumer*.



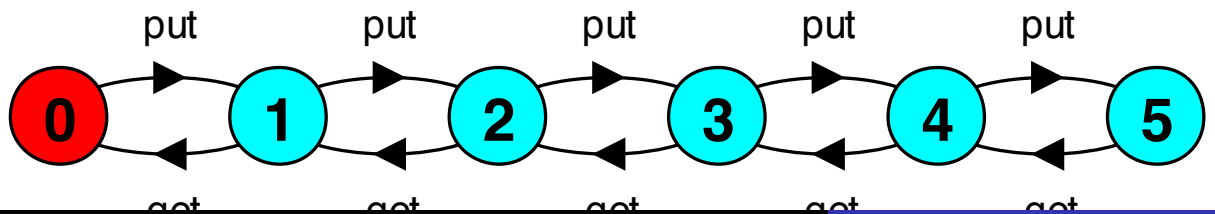
```
BUFFER(N=5) = COUNT[0],
COUNT[i:0..N]
     = (when (i<N) put->COUNT[i+1]
       |when (i>0) get->COUNT[i-1]
       ).
```

- *COUNT* expands to:

$$COUNT[0] = (put \rightarrow COUNT[1])$$
$$COUNT[1] = (put \rightarrow COUNT[2] \mid get \rightarrow COUNT[0])$$
$$COUNT[2] = (put \rightarrow COUNT[3] \mid get \rightarrow COUNT[1])$$
$$COUNT[3] = (put \rightarrow COUNT[4] \mid get \rightarrow COUNT[2])$$
$$COUNT[4] = (put \rightarrow COUNT[5] \mid get \rightarrow COUNT[3])$$
$$COUNT[5] = (get \rightarrow COUNT[4])$$

```
PRODUCER = (put->PRODUCER).
CONSUMER = (get->CONSUMER).

||BOUNDEDBUFFER = (PRODUCER||BUFFER(5)||
CONSUMER).
```



Nested Monitors and Semaphores

- Suppose that, in place of using the *count* variable and condition synchronization directly, we instead use two semaphores *full* and *empty* to reflect the state of the buffer.

```
const Max = 5
range Int = 0..Max

SEMAPHORE ...as before...

BUFFER =  (put -> empty.down ->full.up ->BUFFER
          |get -> full.down ->empty.up ->BUFFER
          ).

PRODUCER = (put -> PRODUCER).
CONSUMER = (get -> CONSUMER).

||BOUNDEDBUFFER = (PRODUCER|| BUFFER || CONSUMER
                  ||empty:SEMAPHORE(5)
                  ||full:SEMAPHORE(0)
                  )@{put,get}.
```

Does this behave as desired?

- **It deadlocks after the trace *get*!!!**
- Why? *CONSUMER* tries to get a character, but the buffer is empty. It blocks and releases the lock on the semaphore *full*. PRODUCER tries to put a character into the buffer, but also blocks.
- It is called *nested monitor problem*.

- Sequences *put* $\rightarrow$ *empty.down* and *get* $\rightarrow$ *full.up* in *BUFFER* are wrong!
  empty.down MUST precede *put* and *full.up* MUST precede *get*, but this *cannot* be done in *BUFFER*!
- Correct Buffer Model:

```
BUFFER =   (put -> BUFFER
           |get -> BUFFER
           ).

PRODUCER =(empty.down->put->full.up->PRODUCER).
CONSUMER =(full.down->get->empty.up->CONSUMER).
```

- The semaphore actions have been moved to the producer and consumer.
- LTS is isomorphic to the previous solution with *COUNT*!

Monitor Invariants

- An invariant for a monitor is an assertion concerning the variables it encapsulates. **It must always hold**.

CarParkControl Invariant:		$0 \leq spaces \leq N$
Semaphore Invariant:		$0 \leq value$
Buffer Invariant:		$0 \leq count \leq size$
	and	$0 \leq in < size$
	and	$0 \leq out < size$
	and	$in = (out + count)$ **modulo** $size$

- Invariants are very helpful in reasoning about correctness of monitors using a logical *proof-based* approach. They are less useful for *model-checking* techniques (but also useful).

- *Passive* vs Active Processes:
  - A process is **active** is it initiates (output) actions.
  - A process is **passive** if it responds to (input) actions. Monitors are passive processes.
- Does nested monitors always lead to errors?
- **'Ask first, do later' principle.**
- The problem with a solution to the bounded buffer problem with semaphores has several roots, and the explanation given in the textbook is incomplete and a little bit misleading.

- Consider the first solution (from page 4) but the a different buffer.

New buffer:

$$BUFFER = (empty.down \rightarrow put \rightarrow full.up \rightarrow BUFFER \mid full.down \rightarrow get \rightarrow empty.up \rightarrow BUFFER)$$

- There is no deadlock with new buffer!

Old buffer:

$$BUFFER = (put \rightarrow empty.down \rightarrow full.up \rightarrow BUFFER \mid get \rightarrow full.down \rightarrow empty.up \rightarrow BUFFER)$$

- System deadlocks after *get* with the old buffer!

- Is anything wrong with this new red buffer and the new solution that use it?
- *BUFFER* is no longer *passive*, so it is not a *monitor*, neither *PRODUCER* nor *CONSUMER* can do anything without *BUFFER* acting first!
- This might be a valid solution in some circumstances, bot not for a standard interpretation of Consumer-Producer problem.
- Note that this new buffer implements 'Ask first, do later' principle!
- New *PRODUCER* and new *CONSUMER*:

$$PRODUCER = (canIproduce \rightarrow put \rightarrow PRODUCER)$$
$$CONSUMER = (canIconsume \rightarrow get \rightarrow CONSUMER)$$

- New composition:

$$\| BOUNDED\_BUFFER = ((PRODUCER \| CONSUMER \| BUFFER \| empty : SEMAPHORE(5) \| full : SEMAPHORE(0)) /\{empty.down/canIproduce, full.down/canIconsume\})@\{put, get\}$$

Full new solution

$$const\ Max = 5$$
$$range\ Int = 0..Max$$
$$SEMAPHORE(N = 0) = SEMA[N]$$
$$SEMA[v : Int] = (up \rightarrow SEMA[v + 1] \mid when(v > 0)down \rightarrow SEMA[v - 1])$$
$$SEMA[Max + 1] = ERROR$$
$$BUFFER = (empty.down \rightarrow put \rightarrow full.up \rightarrow BUFFER \mid$$
$$full.down \rightarrow get \rightarrow empty.up \rightarrow BUFFER)$$
$$PRODUCER = (canIproduce \rightarrow put \rightarrow PRODUCER)$$
$$CONSUMER = (canIconsume \rightarrow get \rightarrow CONSUMER)$$
$$\| BOUNDED\_BUFFER = ((PRODUCER \| CONSUMER \| BUFFER$$
$$\| empty : SEMAPHORE(5) \| full : SEMAPHORE(0))$$
$$/\{empty.down/canIproduce, full.down/canIconsume\})@\{put, get\}$$

- SEMAPHORE and BUFFER are *passive*, i.e. monitors.
- Monitors are *nested* here.
- 'Ask first, do later' principle is used in both BUFFER and *active* processes CONSUMER and PRODUCER.
- The solution works! Its LTS is the same as for correct bounded buffers discussed previously.

- *Nested monitors* increase chances of errors, but do not cause errors when used carefully. Similarly like "goto" increases chances of errors, but not all programs with "goto" are wrong.
- 'Ask first, do later' principle. Is is always necessary?
- Consider the solution where BUFFER and SEMAPHORE are not nested, but we have another new PRODUCER and another new CONSUMER.

$$
\begin{aligned}
&const\ Max = 5\\
&range\ Int = 0..Max\\
&SEMAPHORE(N = 0) = SEMA[N]\\
&SEMA[v : Int] = (up \rightarrow SEMA[v+1] \mid when(v > 0) down \rightarrow SEMA[v-1])\\
&SEMA[Max+1] = ERROR\\
&BUFFER = (put \rightarrow BUFFER \mid get \rightarrow BUFFER)\\
&PRODUCER = (put \rightarrow empty.down \rightarrow full.up \rightarrow PRODUCER)\\
&CONSUMER = (get \rightarrow full.down \rightarrow empty.up \rightarrow CONSUMER)\\
&\| BOUNDED\_BUFFER = (PRODUCER \| CONSUMER \| BUFFER\\
&\qquad \| empty : SEMAPHORE(5) \| full : SEMAPHORE(0))@\{put, get\}.
\end{aligned}
$$

- No deadlock!
- No nested monitors and **NO** 'Ask first, do later' principle!

- The last solution looks formally OK, but what the sequence $get \rightarrow put$ means when the buffer is empty?
- It still may be a valid solution in some peculiar circumstances, but most likely it will be a serious error that could produce some random values and could be difficult to detect.
- The solution is **not** equivalent to the other ones as the trace *get* is valid. LTS is show it, but the system will not deadlock.
- On a 'syntax level', i.e. without interpreting actions, this is a valid specification!