Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

with thanks to Mike Brady

Producers and Consumers

- Essentially a *pipeline* of separate sequential programs.
  - E.g. concatenation of unix commands.
- Programs communicate via buffers.
  - Implemented in different ways, but, e.g.
    - Shared memory, flags, semaphores, etc.
    - Message passing over a network
- Flow of data is essentially one-way.

Imagine a Situation…

- A ‘Consumer’ is waiting for a buffer to become non-empty to take an item from it, decrementing its counter.
- A ‘Producer’ adds items to the buffer from time to time, incrementing its counter.
- We could use a mutex to control access to the counter.
- How do we organise the consumer?

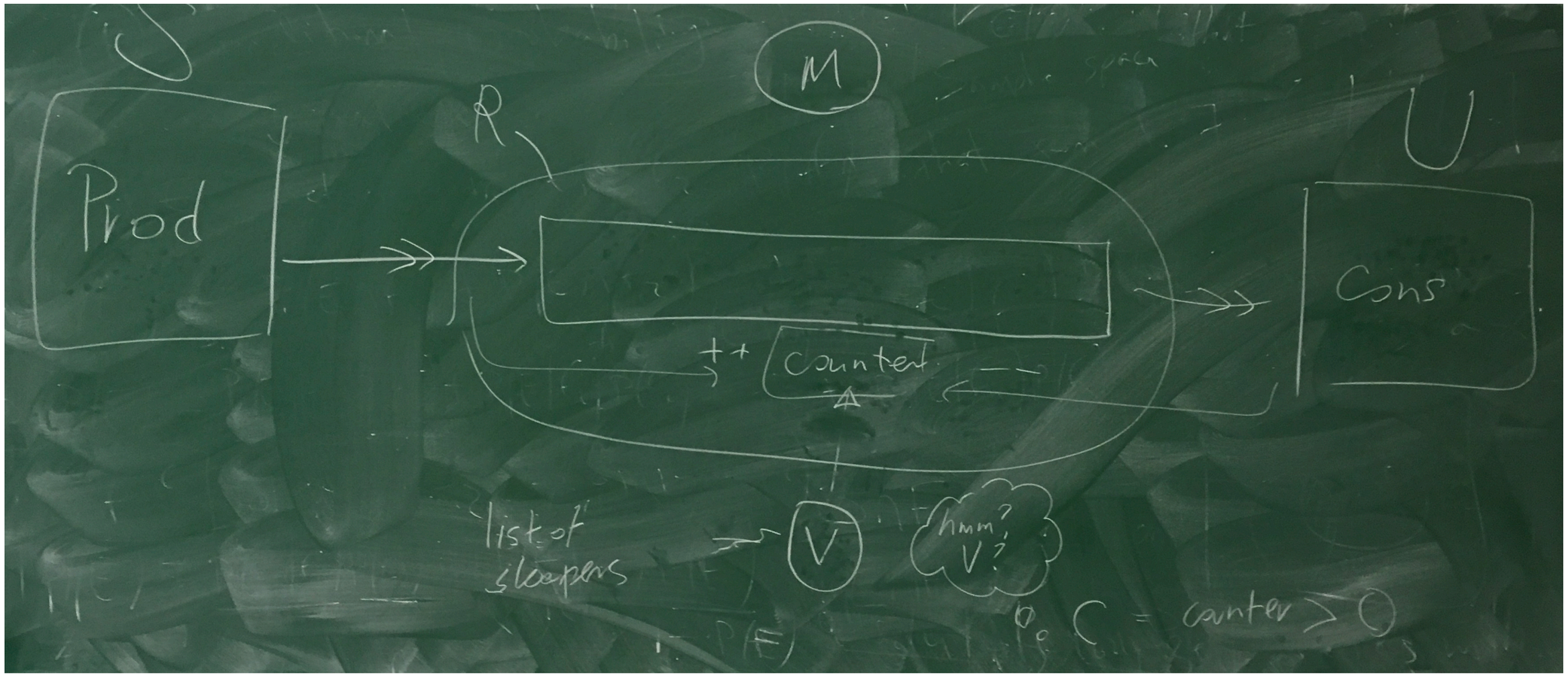
Condition Variables

- A generalised synchronisation construct.
- Allows you to acquire mutex lock when a *condition* relying on a shared variable is true and to sleep otherwise.
- The ‘condition variable’ is used to manage the mutex lock and the signalling.
- Slightly tricky.

Condition Variables (2)

- Dramatis Personae:
  - A mutex variable M,
  - A shared resource R to be guarded by M,
  - A condition variable V,
  - A condition C,
  - A thread U that wishes to use R, protected by M, on condition that C is true,
  - A thread S that will signal V, presumably when it has done something that might indirectly change the value of C.

(U,S) = (Consumer,Producer)



From Thread U's POV (1)

- Acquire Mutex M
  - This controls access to R,
  - but also must control access to any shared variables that C depends on.
- If C is true, continue – the mutex is available unfettered,
  - i.e. if C is true, the mutex can be used as normal.
- if C is false…

From Thread U's POV (2)

- If C is false, wait for condition variable V to be *signalled*.
- This is tricky, as access to R is controlled by M.
  - So, Mutex M is unlocked,
  - Thread sleeps, to be woken when V is signalled,
  - Then, M is re-acquired.
    - No guarantee you'll get it right away–maybe another thread will.
- So now, if V has been signalled…

From Thread U's POV (3)

- Since V has been signalled, there is a chance that the condition C is now true.
- Re-evaluate C:
  - If true, then the mutex is available for you to use,
  - If false, back to previous page.
- Our 'if' needs to be replaced by a 'while'…

From Thread U's POV (4)

- Acquire Mutex M
- While !C
  - Unlock M
  - Wait for V to be signalled
  - Lock M
- Continue
- Unlock M (finished)

```
pthread_lock(&m)

while (!C)

   pthread_cond_wait(&v,&m)
```

or

```
   pthread_cond_timedwait(&v,&m,<time>)

continue…

pthread_unlock(&m)
```

Thread S (1)

- Thread U is the 'user' of the condition variable V.
- If the condition is not true, U unlocks V's mutex M and sleeps, waiting for some other thread S to signal V and thereby waking U to check the condition again.

Thread S (2)

- Acquire Mutex M
- Do something
- Unlock Mutex M
- Signal the condition variable
  - This will awaken a thread that is waiting on the condition variable.

```
pthread_lock(&m)

Do something…

pthread_unlock(&m)

pthread_cond_signal(&v)
```

or

```
pthread_cond_broadcast(&v)
```