Producer-Consumer Problem

Joseph Kehoe

Producer Consumer

- A common pattern is for the division of labor amongst threads
  - E.g. Some threads consume while others produce
  - Producers create items of some kind and add them to a data structure (buffer)
  - Consumers remove the items and process them
  - E.g. Event driven programs
    - Consumers known as event handlers

Producer – Consumer Pattern

- While an item is being add to or removed from buffer the buffer is in an inconsistent state
  - Therefore we must guarantee exclusive access to the buffer
- If a consumer thread arrives when the buffer is empty it must wait until a producer adds a new item
- Producer
  - Event= createEvent()
  - Buffer.add(event)
- Consumer
  - Event = Buffer.get()
  - Event.process()

Hint (part 1)

- Mutex=semaphore(1) //control access to buffer
- Items=semaphore(0) //blocks when buffer is empty

Hint (part 2)

- Local variable event (for adding to or taking from buffer)
  - Event is local to thread
  - Each thread has their own version of event!
    - Each thread may have their own run-time stack so all local variables are thread specific
    - If threads are objects then we can add attributes to the objects
    - If threads have unique ID’s then we can use ID as an index into an array or hash table

Solution

- Producer
  - Event= createEvent()
  - Mutex.wait()
  - Buffer.add(event)
  - Items.signal()
  - Mutex.signal()
- Consumer
  - Items.wait()
  - Mutex.wait()
  - Event = buffer.get()
  - Mutex.signal()
  - Event.process()

Improved Solution

- Signaling inside the mutex can be inefficient
  - Why?
- Improved Producer
  - Event= createEvent()
  - Mutex.wait()
  - Buffer.add(event)
  - Mutex.signal()
  - Items.signal()

Incorrect Solution

- Items can be inaccurate given certain interleavings
  - We can try correct this…
- Consumer
  - Mutex.wait()
  - Items.wait()
  - Event = buffer.get()
  - Mutex.signal()
  - Event.process()

Producer-Consumer with finite buffer

- If buffer is finite it can fill up
- In that case producers should wait until the buffer has freed up some space before adding to buffer
- We cannot check the value of items as we are not allowed to do this!
- Hint
  - Add another semaphore initialised to the buffer size!
  - Spaces=semaphore(buffer.size)

Solution

- Producer
  - Event= createEvent()
  - Spaces.wait()
  - Mutex.wait()
  - Buffer.add(event)
  - Mutex.signal()
  - Items.signal()
- Consumer
  - Items.wait()
  - Mutex.wait()
  - Event = buffer.get()
  - Mutex.signal()
  - Spaces.signal()
  - Event.process()