CS533 Concepts of Operating Systems

Class 4

On The Duality of Operating System Structures
The basic idea

- There are two common ways to structure systems:
  - message oriented (i.e., event based)
  - procedure oriented (i.e., multi-threaded)

- They are duals of each other
  - any program written in one form can be automatically transformed into the other form with no loss of functionality

- Don’t waste your time arguing about which one is inherently better
Message oriented systems

- Small number of long-lived processes (or threads)
- Communication among the processes only via message channels/queues
  - each process extras message requests from its queue and processes them before extracting the next request
  - no shared data
- Essentially the same as event-based programming
  - no shared data, no need to synchronize access to shared data
  - reactive execution model using handlers
  - each request is executed to completion
Procedure oriented systems

- Globally shared data with accesses from multiple threads synchronized using monitors (locks and condition variables)
- Process/thread per task model
  - threads may be short-lived (created by fork to handle a task)
  - they can wander all over the system and access any data
  - communication among threads is via the shared data
The duality mappings

- Process == monitor
- Message channel == monitor entry method
- send message, wait for reply == entry method call
- asynchronous send, work, wait == fork, work, join
- send reply == return
- main loop, wait for message == lock, entry method
- wait for message == wait on condition variable
- send message == signal condition
The duality

- One form can be converted into the other via a straightforward program transformation
- The logic of the program itself is unchanged
- The structure of the program is unchanged
- Any visual difference is purely syntactic
  - computation is triggered by `SendMessage` in one model and `Fork` or `Signal` in the other
  - computation is blocked by `AwaitReply` or `WaitForMessages` in one model and by `Join` or `WaitCondition` in the other
  - data access is serialized by the `WaitForMessages` loop in one model and by a `Monitor Mutex` in the other
Conclusion

- Disagreement between the two sides seems to be based more on emotion than fact
  - there really isn’t a fundamental difference between these two models

- So how should you decide which model to use?
  - maybe a particular architecture has better hardware support for one model than the other
  - i.e., which one is better depends on the platform you are running on