On the Duality of OS Structures

Mark Sutherland*

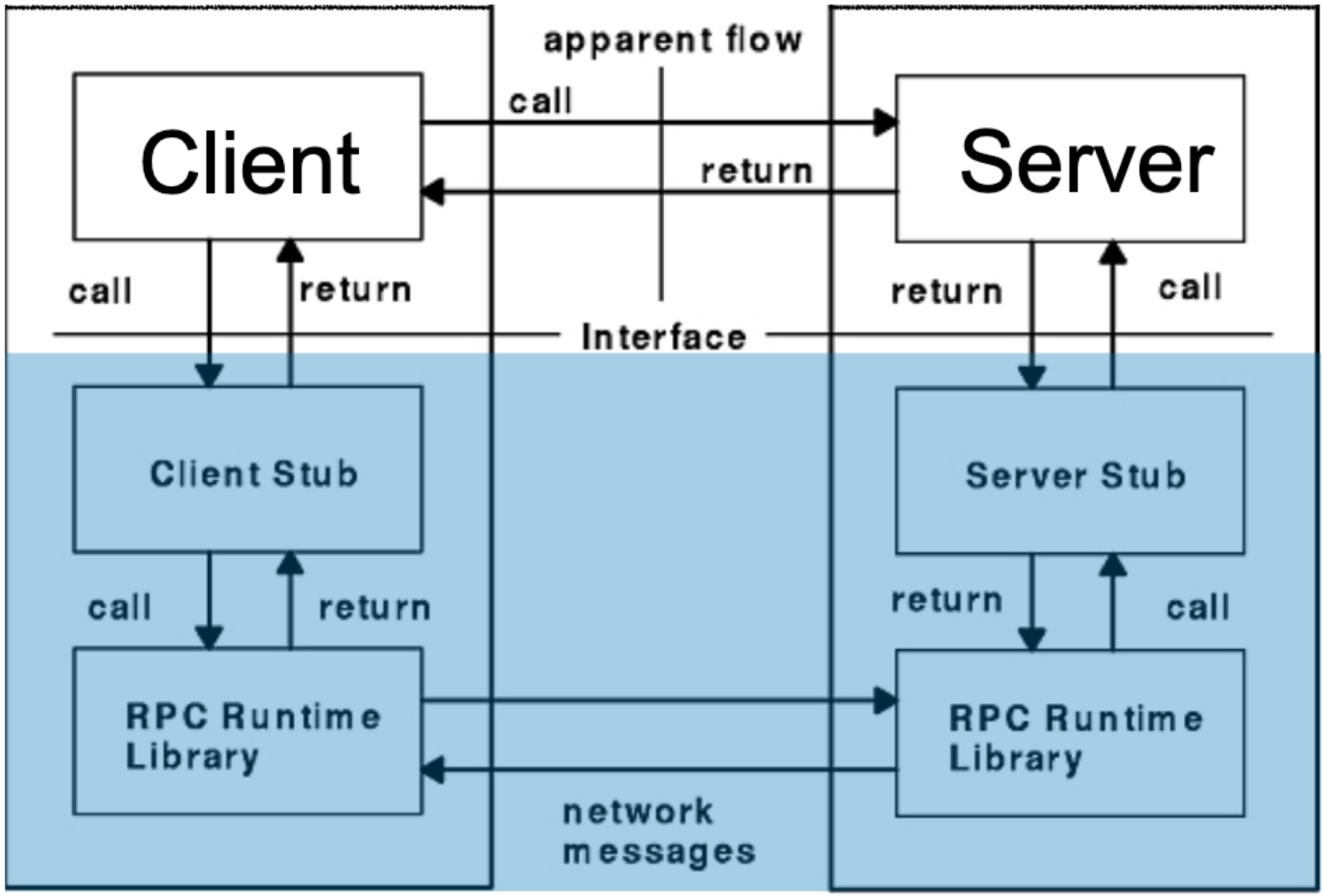
15. 10. 2020

*Based on slides by Marios Kogias & Rishabh Iyer

Paper Recap

- Two rough models of OS designs
  - Message-oriented
  - Procedure-oriented
- Two models are duals
  - Program in one model has direct counterpart (dual) in other
- Dual programs:
  - Are logically identical
  - Can be implemented to have similar performance

Application's Perspective



Can applications be agnostic to what's contained in the blue box?

How is this relevant to modularity?

- Message Passing enforces modularity
  - All communication via explicit messages
  - Modules are isolated
  - Propagation of errors is reduced
- What other components of the stack are modular?
  - For message-oriented? For procedural?

What might they have missed?

- Claim: the two can have equal performance given equivalent scheduling
  - Brainstorm: what might not be sufficiently discussed?

- How might modern systems make this relevant again?
  - CPU performance growth since 1979
  - Criticality of (re)scheduling operations
  - Applications don't often use the same data formats. Need a common representation for messages

Designing good interfaces

- We discussed how server interfaces are defined (IDLs)
- Design considerations for message passing systems:
  - How do I name processes I want to communicate with?
  - What is the message format?
  - Semantics of asynchronous operations?

Further Interesting Reading

- The Microkernel (Barrelfish) – SOSP'09
- LegoOS – OSDI'18
- Snap: Microkernels for Networking – SOSP'19

Last year’s slides…

Duality

Message-oriented system	*Procedure-oriented system*
Processes, **CreateProcess**	Monitors, NEW/START
Message Channels	External Procedure identifiers
Message Ports	ENTRY procedure identifiers
SendMessage; AwaitReply (immediate)	simple procedure call
SendMessage; . . . AwaitReply (delayed)	FORK; . . . JOIN
SendReply	RETURN (from procedure)
main loop of standard resource manager, **WaitForMessage** statement, **case** statement	monitor lock, ENTRY attribute
arms of the **case** statement	ENTRY procedure declarations
selective waiting for messages	condition variables, WAIT, SIGNAL

Can be thought of as duality of IPC mechanisms

Shared Memory vs Message Passing

- Decades-old debate on the right IPC mechanism
  - Have also been proven to be duals
- Shared memory:
  - Writes to local memory/registers are globally visible
  - Communication is implicit
- Message passing:
  - Communication must be explicitly specified.
  - Must communicate with a process to share data with it