

On the Duality of Operating System Structures

Hugh Lauer
Xerox Corporation
Roger Needham
Cambridge University

Presented by Elizabeth Keniston
CS 533 - Winter 2009

Basic Idea



- Most operating systems can be divided into one of two categories:
 - Message-oriented
 - Procedure-oriented
- For each operating system in one category it's possible to create a “dual”, or direct counterpart using the other system.
- Neither system is inherently better than the other.

Goals of the paper



- Present the authors' observations about operating systems
- Describe in detail two canonical models that they have observed
- Show that the two models are duals of each other
- Some evidence is presented, but no attempt is made to rigorously prove their assertions.

Some historical perspective



- This paper was written in 1978.
- UNIX was becoming fairly well-known.
- IBM's OS/360 had been around for several years.
- There was a wider variety of types of operating systems than there are today.
 - Today, many operating systems are “related”, or based on the same operating system.

A note on the two models



- Message-oriented systems are similar to event-based systems
- Procedure-oriented systems are similar to thread-based systems
- The event vs. thread arguments were already heated in 1978!

Real operating systems



- No real operating system exactly fits either model in all respects.
- Many operating systems have some subsystems that fit one model, and others that fit the other.
- Most systems that don't fit either model and can't be divided into subsystems that fit either model are “ill-structured and unstable... unreliable, unmanageable, uneconomic, and unusable.” [3]

Message-oriented system



- Characterized by efficient message passing between processes
- Convenient operations for sending and waiting for messages, waiting for a particular kind of message, and checking the message queue
- Fewer processes
 - Number of processes and connections between processes are more static

Message-oriented system



- Most processes directly associated with a system resource
- Congested resources result in processes blocking while waiting for replies to their messages
- Very little shared memory
 - Data is passed by reference in messages
 - Processes never touch such data except after receiving it in a message, but before sending it off in another message.

Procedure-oriented system




- Many lightweight procedures (basically threads); can be easily created and destroyed
- Resources handled with shared data
- Data is protected with various forms of locks, semaphores, monitors, etc.
- Little to no direct communication between processes
- Congested resources result in processes blocking while waiting on monitor locks or condition variables

Monitors



- A monitor is a special kind of module that has private data and procedures, protected with a lock.
- Processes must acquire the lock when they call an entry procedure (a procedure which can be called from outside the monitor).
- Only one process can operate inside the monitor at a time.

Duality



- The authors claim that the two models are “duals” of each other
 1. A system constructed with the primitives defined by one model can be mapped directly into a dual system, which fits the other model.
 2. Dual programs are logically equivalent.
 3. The performance of a system in one model is can be made as efficient as its dual of the other model.

Two resource managers



Message-oriented:

```
begin m: messageBody;
  i: messageld;
  p: portid;
  s: set of portid;
  ... -local data and state information for this process
  initialize;
  do forever;
    [m, i, p]← WaitForMessage[s];
    case p of
      port1 =>...; -algorithm for port1
      port2 =>...
        if resourceExhausted then
          s *- s - port2;
          SendReply[i, reply];
          ...; -algorithm for port2
      portk =>...
        s *- s + port2 ...; -algorithm for portk
    endcase;
  endloop;
end.
```

Two resource managers



Procedure-oriented:

```
ResourceManager: MONITOR =
  C: CONDITION;
  resourceExhausted: BOOLEAN;
  ... "global data and state information for this process
  proc1 : ENTRY PROCEDURE[ . . . ] =
    ...; "algorithm for proc1
  proc2: ENTRY PROCEDURE[ . . . ] RETURNS[ . . . ] = BEGIN
    IF resourceExhausted THEN WAIT c;
    RETURN[results];
    ...
    END; "algorithm for proc2
  procL: ENTRY PROCEDURE[ . . . ] = BEGIN
    resourceExhausted«- FALSE;
    SIGNAL C;
    ... >
    END; "algorithm for procL
  endloop;
  initialize;
END.
```

Duality mapping



Message-oriented

Process, CreateProcess

Procedure-oriented

Monitors, NEW/START External



Duality mapping



Message-oriented

Process, CreateProcess

Message channels/ports



Procedure-oriented




Monitors, NEW/START External

Procedure identifiers ENTRY

Duality mapping



Message-oriented

Process, CreateProcess 
Message channels/ports 
SendMessage/AwaitReply 
(immediate)

Procedure-oriented

Monitors, NEW/START External
Procedure identifiers ENTRY
Procedure call

Duality mapping



Message-oriented

Process, CreateProcess

Message channels/ports

SendMessage/AwaitReply
(immediate)

SendMessage/AwaitReply
(delayed)



Procedure-oriented

Monitors, NEW/START External

Procedure identifiers ENTRY

Procedure call

FORK/JOIN

Duality mapping



Message-oriented

Process, CreateProcess 

Message channels/ports 

SendMessage/AwaitReply
(immediate) 

SendMessage/AwaitReply
(delayed) 

SendReply 

Procedure-oriented

Monitors, NEW/START External

Procedure identifiers ENTRY

Procedure call

FORK/JOIN

RETURN (from procedure)
monitor

Duality mapping

Message-oriented


Process, CreateProcess 

Message channels/ports 

SendMessage/AwaitReply
(immediate) 

SendMessage/AwaitReply
(delayed) 

SendReply 

Main loop of standard
resource manager, WaitFor
Message statement, case
statement 

Procedure-oriented

Monitors, NEW/START External

Procedure identifiers ENTRY

Procedure call

FORK/JOIN

RETURN (from procedure)
monitor

Lock, ENTRY attribute

Duality mapping

Message-oriented


Process, CreateProcess 

Message channels/ports 

SendMessage/AwaitReply
(immediate) 

SendMessage/AwaitReply
(delayed) 

SendReply 

Main loop of standard
resource manager, WaitFor
Message statement, case
statement 

Arms of the case statement 

Procedure-oriented

Monitors, NEW/START External

Procedure identifiers ENTRY

Procedure call

FORK/JOIN

RETURN (from procedure)
monitor

Lock, ENTRY attribute

ENTRY procedure declarations

Duality mapping

Message-oriented


Process, CreateProcess 

Message channels/ports 

SendMessage/AwaitReply
(immediate) 

SendMessage/AwaitReply
(delayed) 

SendReply 

Main loop of standard
resource manager, WaitFor
Message statement, case
statement 

Arms of the case statement 

Waiting for messages 

Procedure-oriented

Monitors, NEW/START External

Procedure identifiers ENTRY

Procedure call

FORK/JOIN

RETURN (from procedure)
monitor

Lock, ENTRY attribute

ENTRY procedure declarations

Condition variables, WAIT,
SIGNAL

Similarity of programs



- A system constructed with the primitives defined by one model can be mapped directly into a dual system, which fits the other model.
- A client program written for one system can be transformed for the other system by replacing the primitives from the first model with the primitives of the other.
- This does not affect the logic of the client program
 - None of the important parts are touched or rearranged
 - The semantic component is invariant

Preservation of performance



- Three important elements that affect performance
 1. The execution times of the client programs themselves
 2. The computational overhead of the system calls made by the program
 3. The queuing and waiting times caused by shared resources, dependence on external events, and scheduling decisions

Program execution times



- The duality transformation doesn't modify the main bodies of the programs
- Algorithms compute at same speed
- Same number of additions, multiplications, comparisons, etc.
- Therefore, this component will take the same amount of computing power.

System calls/System resources



- The authors “assert without proof” that the duals can be made to execute as efficiently as the corresponding facilities in the other model.
 - Sending a message vs. calling/forking to an ENTRY procedure
 - Allocate memory, queue, and context switch
 - Wait for new messages vs. leaving a monitor
 - Unqueue message vs. unqueue waiting process
 - Process switching can be made equally fast

System calls/System resources



- Scheduling can be made identical in each version
 - Queuing/dequeuing implementation can be equivalent
 - Message queues vs. process queues
 - Context switches can occur at the same times
 - Each system can have similar scheduling responses to external events, kernel operations, etc.
 - Therefore, events happen in the same order.

System calls/System resources



- Not much detail is included about these equivalencies – we are left to believe or not as we choose.
- One example is cited: the GEC 4080
 - Implemented with message queuing, process switching and dispatching as fast operations
 - An implementation of the Mesa system that uses the dual operations was found to operate with similar speed

Preservation of performance



- Therefore, the authors assert that the total lifetime of computation is the same for both models.
- However, it isn't easy to change the structure of most OS's.
- One citable case: Cambridge CAP computer
 - Originally message-oriented
 - Switched to process-oriented, with little change

Underlying differences



- No inherent difference
- Client systems have similar program structures (0th order consideration)
- Computation complexity is similar in each system (1st order consideration)
- Therefore, any reason to choose one system over the other must be two or more steps removed from the primary consideration of the designer.

Underlying differences



- Machine architecture should be the main reason to choose one or the other.
 - Example: if it is easy to allocate message blocks and queue messages, but difficult to build a protected procedure call mechanism – use the message oriented system.
 - Some other factors
 - Organization of real/virtual memory
 - Size of the stateword which must be saved on every context switch
 - Scheduling overhead

Conclusion



- Event-based vs. thread-based arguments have been going on for over 30 years.
- The authors want everyone to just get along and stop arguing – both systems are valid!