Federated Systems

Lecture 8
Operating System Structure
Rich Martin
October 26th 1999

Today

Kernel Structure for Federated Systems

- Unix, Thompson and Ritchie
- QNX
- Inferno

OS architecture and design

- What are the key abstractions of these OSes
- How do they map to real machine resources?
- What are the performance and fault characteristics of these designs?

Today (cont)

Common Themes

- Cooperating processes
  - How to connect them together
  - Name spaces to resource mappings
- OS structures
  - Processes, messages, namespaces
  - Mapping to a federated architecture

Unix

Abstractions

- Processes
- Pipes
- Hierarchical namespaces (Multics lite)
- Virtual memory (swapping)
- Unstructured files
- No special programs
- No special languages

Process

- address space
  - Segments (stack, heap, bss, etc)
- thread of control
  - one per process
  - machine register set (PC, a–regs, d–regs)
  - Machine registers not well–defined in unix+C
  - Allows ports to many machines

Pipes

How do processes send data to each other?

- Via the filesystem?
- via shared memory?

Pipe, like a file, an ordered, and unstructured

- byte stream vs. byte array

Pipes (cont)

Pros:

- Simple abstractions
- Easy to map to different communication hardware (e.g. ethernet, serial lines)

Too unstructured?

- Forces user to manage all details of communication

Basis of sockets

Address spaces

Process abstraction

- Fork(), exec()
  - compare with spawn() in QNX
- Easy to clean up resources on failure
- Harder to build abstraction
  - Compare with a Dis/Java VM
  - Even with MMU support, a pain
    - Look at x86 memory management hardware

Namespaces

Multics–style hierarchical namespace

- Modern unixes have multiple namespaces
  - (e.g. so an ipcs)

Alternatives today?

Q: Fundamental part of OS?

- vs. QNX, which only has process namespace?
- as in plan 9/inferno, where everything is a file?

Command Shells

Uniform programming interface

- Via system call interface
- few special programs (init+ swapper)
- Command shells are just programs

Unix Fault Model

Faults in address spaces

- Halt the AS
- clean up hernel resources:
  - Memory
  - Files descriptors and state (e.g. sockets)
  - Shared memory? Semaphores?

What about a fault in the executive?

QNX

Micro–kernel design

- Word overused, what does a µ–kernel mean?

OS abstractions

- processes, like unix
- messages, unlike unix
- networked PID space, unlike unix, inferno
- No filesystem namespace or FS in the kernel

QNX namespace

What belongs in the executive vs as an extension?

- address spaces
- messages–passing (restricted)
  - proxies
- signals
- scheduling

Process namespace

All processes are known by node:number

- Symbolic names to PIDs?
- A well known process must serve as a PID nameserver
- Similar issues as with the filesystem
  - e.g. /foo/bar/baz => inode X

QNX Messages

Message passing with restrictions

- Request/Reply model of message passing
- process scheduling tied to messaging
- Very simple abstraction
- Performance?

QNX Messages (cont)

Process A	Process B
`send(req_buf, reply_buf)`	
Sends request, blocks →	`Receive()`
	If no msg, block else receive
←	`Reply()`
Fill in reply buffer unblock process	

Message Proxy

What if you can't wait?

- Idea: send a message to a proxy
- Proxy sends a pre−computed message to the recipient
- main process can continue
- what if you can't pre−compute the message?

Messaging & Scheduling

How to not copy data when sending?

- block sending process!
- Ties scheduling to messaging

- Q: How to form a smooth pipeline?
  - Must use a proxy
  - or Multiple processes

QNX Signals

How to force control change as well as data?

- send a signal
- Signal changes control of the recipient
  - Software interrupt
- Later versions of unix have expanded signal capabilities

Interaction of signals and messages?

- signals in blocked states cause aborted messages

Interrupt Handers

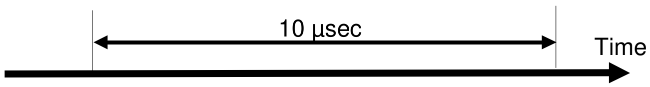
An interrupt occures. How long to unwind state to the the target process?

Three kinds of latency:

- From data ready to kernel ready
- Through kernel to handler
- From handler to user process

Simple Interrupt Experiment

```
Lanai polls clock
Lanai pulls IRQ
```




```
IRQ
Jump to IRQ handler
jump to device handler
```

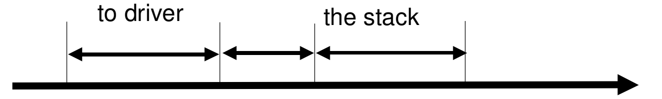
Sun Ultra–1
Myricom LANai 4.1

```
Driver polls lanai clock
Driver stores clock
```


Going through the scheduler

OS code
to driver

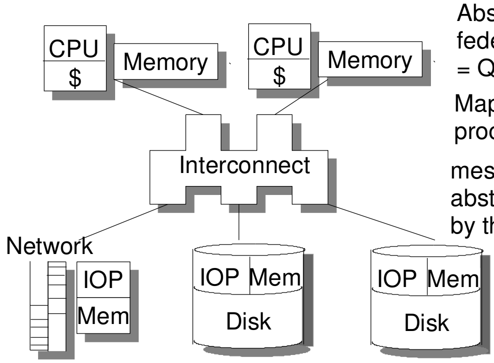
OS "unwinds"
the stack



Driver issues
signal

Run process
Deliver signal

QNX to federated architecture

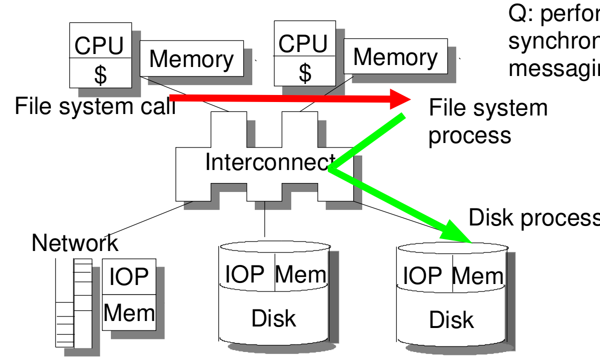


Abstraction of a federated machine = QNX system?

Map µ–kernels to all processors in system

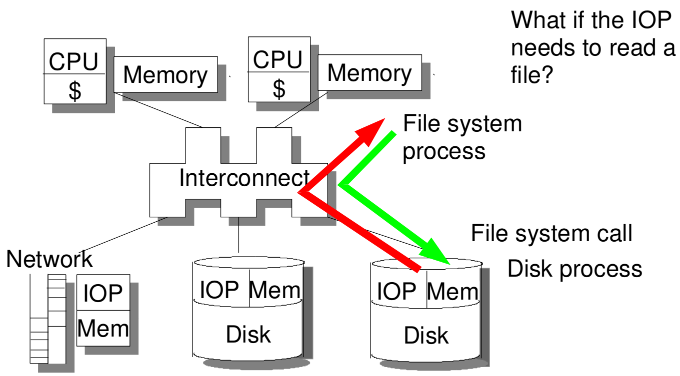
messaging, scheduling abstractions provided for by the system

QNX to federated architecture



Q: performance of synchronous messaging?

QNX to federated architecture



Device performance?

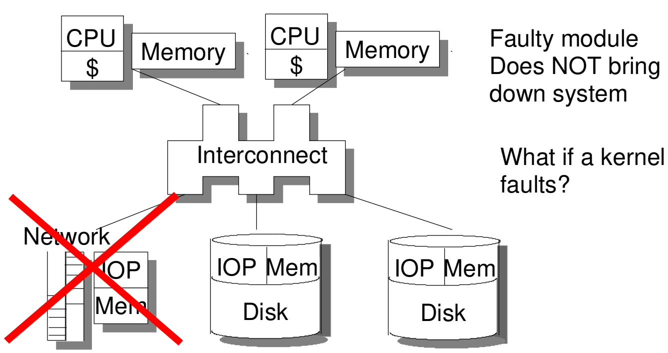
Nodes must possess an MMU?

- How to implement process abstraction w/o?
- Are different ISA's a problem?
  - endian problems?

What about shared memory?

- Mmap, sysV shared memory?

Fault Models



μ–kernel right abstraction?

Shared memory is harder in hardware

Messaging is easier in hardware

- Betrer match to hardware resources?
- Harder to lay out data structures
- easier to pass control

Pipes vs. messaging

Pipes vs. Messages

Pipe is an ordered sequence of bytes

Messaging: sequence of atomic data

- ordered or unordered?

Pipes more generic but also

- heavier burden on higher layers
- Wasting resources, hardware already moves data in message units

Pipes vs. Messaging

Pipe logic requires a select of all pipes.
How to manage N–to–1 queue abstraction?
Logic to reconstruct incomming messages do to
byte stream abstraction,

```
while (msg_not_all_here) {
   any = select(FD_READ, ...  );
   next_fd = fd_low;
   while (FD_ISSET(next_fd)) {
      bytes[fd] += read(fd,buffer[next_fd],maxlen);
      if (bytes[fd] > minMsgSize ) {
         if (messages passed checks ...) {
         }
   }
}
```


Messaging vs. Pipes

reconstruction

merging (n–to–1 queue semantics)

ordering and loss semantics?

```
if (msgid = recv(buf, len) ) {
   check message
   do processing
}
```


Inferno

Touted as network infrastructure OS

Main Abstractions

- Single namespace
  - standard read/write calls translated into messaging
- Typesafe language (Limbo)
  - Channel (pipe?) as a primitive datatype
- Virtual Machine (Dis)

Language/VM based OS

Language provides protection, Not AS

- E.g. Spin OS

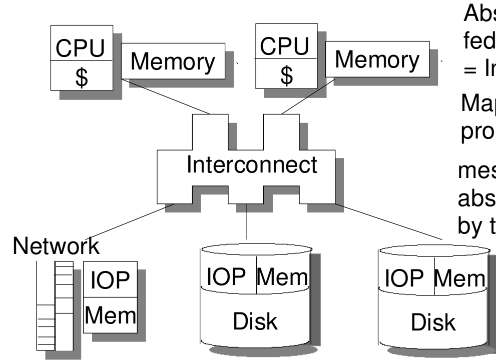
VM based OS uses a Virtual Machine to provide protection

- Java, Dis
- Multiple languages to the VM?

How to reclaim resources from a failed thread?

- (e.g. sockets,time)

Inferno to federated arch.



Abstraction of a federated machine = Inferno system?

Map Dis VM to all processors in system

messaging, namespace abstractions provided for by the system