OS Avoidance
Which OS?

Heavyweight

Lightweight
Heavyweight OS Features

• Separate address spaces
  – virtual memory
• System calls
  – user/privileged-mode distinction
# Weight

<table>
<thead>
<tr>
<th></th>
<th>API call</th>
<th>Thread yield</th>
<th>Message ping/pong</th>
<th>Process creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singularity</td>
<td>80</td>
<td>365</td>
<td>1,040</td>
<td>388,000</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>878</td>
<td>911</td>
<td>13,300</td>
<td>1,030,000</td>
</tr>
<tr>
<td>Linux</td>
<td>437</td>
<td>906</td>
<td>5,800</td>
<td>719,000</td>
</tr>
<tr>
<td>Windows</td>
<td>627</td>
<td>753</td>
<td>6,340</td>
<td>5,380,000</td>
</tr>
</tbody>
</table>
Shedding Weight …

• Software-isolated processes (SIPs)
  – use type safety and memory safety to isolate processes
  – all processes run in same address space
  – all run in privileged mode

• IPC via “contract-based channels”
  – bi-directional, reliable message conduit with exactly two endpoints
  – one thread per endpoint
  – formally specified interaction “contract”
  – no other IPC mechanism
  – act as capability mechanism
Channels Between Network Driver and Network Stack

NetStack

NIC Driver

NicDevice

NicEvents
NIC Driver Contract (1)

```cpp
contract NicDevice {
    out message DeviceInfo(...);
    in message
        RegisterForEvents(
            NicEvents.Exp:READY c);
    in message
        SetParameters(...);
    out message
        InvalidParameters(...);
    out message Success();
    in message StartIO();
    in message ConfigureIO();
    in message
        PacketForReceive(
            byte[] in ExHeap p);
    out message BadPacketSize(
            byte[] in ExHeap p, int m);
    in message
        GetReceivedPacket();
    out message ReceivedPacket(
            Packet * in ExHeap p);
    out message NoPacket();
}
```
NIC Driver Contract (2)

\begin{verbatim}
state START: one {
    DeviceInfo! →
        IO_CONFIGURE_BEGIN;
}

state IO_CONFIGURE_BEGIN: one {
    RegisterForEvents? →
        SetParameters? →
            IO_CONFIGURE_ACK;
}

state IO_CONFIGURE_ACK: one {
    InvalidParameters! →
        IO_CONFIGURE_BEGIN;
    Success! → IO_CONFIGURED;
}

state IO_CONFIGURED: one {
    StartIO? → IO_RUNNING;
    ConfigureIO? →
        IO_CONFIGURE_BEGIN;
}

state IO_RUNNING: one {
    PacketForReceive? →
        (Success!
            or BadPacketSize!)
            → IO_RUNNING;
    GetReceivedPacket? →
        (ReceivedPacket!
            or NoPacket!)
            → IO_RUNNING;
    ...
}
\end{verbatim}
NIC Device Events Contract

```java
contract NicEvents {
    enum NicEventType {
        NoEvent, ReceiveEvent, TransmitEvent, LinkEvent
    }
    out message NicEvent(NicEventType e);
    in message AckEvent();
    state READY: one {
        NicEvent! → AckEvent? !READY;
    }
}
```
Manifest

• Each program has a *manifest*
  – details
    - code resources
    - system resources
    - desired capabilities
    - dependencies on other programs
Scenario

I/O-intensive app

File system

Disk driver

Kernel
Costs

Unsafe Code Tax

No runtime checks: -4.7%
Physical Memory: +6.3%
Add 4KB Pages: +18.9%
Add Separate Domain: +33.0%
Add Ring 3: +37.7%
Full Microkernel:
You’ll Soon Finish 167/169 ...

• You might
  – celebrate
  – take another course
    - 138
    - 160
    - 165
    - 166
    - 168
    - 176
  – graduate (!)
  – do some systems research
  – become a 167/169 TA
Well, not quite …

The final exam is on Friday, May 12 at 2pm in Kassar Foxboro Auditorium.

Closed book; covers the entire course.

Help session 5pm Wednesday, 5/10, in TBD.

Old finals will soon be on the web page.