Operating Systems, Composite, and Future Systems

Gabriel Parmer
aka. Gabe
gparmer@gwu.edu
The George Washington University (GWU)

Advanced Operating and Distributed Systems
What is the class about? Aren’t OSes solved?

1. Exposure to a complex system focused on resource management
2. Cutting edge systems: new paradigms, new ways of thinking

- Topics: Parallelism
  - since 2005 → multicore
- Topics: Reliability
  - increasing code complexity, smaller processors
  - how do we cope with failure?
- Topics: Memory Management
  - increasingly important resource
Embedded Systems Run the World
Focus:

- Predictability – results within a known amount of time → Real-Time
- Reliability – resilience to failure, fault tolerance

...not...

- Efficiency – real fast, transactions/sec
- Parallelism – not worth cost/weight
Focus:
- Efficiency – effective use of hardware
- Parallelism – multi-core renaissance

...not...
- Predictability – target: low average response time
- Reliability – trust the DB, client requests can fail
Embedded Systems $\xrightarrow{\text{convergence}}$ Backend Systems
Embedded + Backend Convergence

Embedded Systems $\leadsto$ Backend Systems

Increasing *complexity of embedded systems*

$\rightarrow$ Efficiency + Parallelism
Embedded + Backend Convergence

Increasing complexity of embedded systems

→ Efficiency + Parallelism

Increasing need for trusted backend systems

→ Predictability + Reliability
Cake – having *and* eating?

- **Reliability** vs. *Predictability*
  - vs. Efficiency
  - vs. Scalability

- **Efficiency** vs. *Scalability*
  - vs. Predictability
  - vs. Reliability

- **Scalability** vs. *Predictability*
  - vs. Efficiency
  - vs. Reliability

- **Predictability** vs. *Efficiency*
  - vs. Scalability
  - vs. Reliability
Cake – having and eating II

- Reliability
- Predictability
- Efficiency
- Parallelism
Embedded + Backend goals have converged...
...but workloads still very different

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>RAM</th>
<th>CPU</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Embedded</td>
<td>Low</td>
<td>Slow</td>
<td>simple RT tasks</td>
</tr>
<tr>
<td>2</td>
<td>Embedded</td>
<td>Med</td>
<td>Med</td>
<td>simple RT + general tasks</td>
</tr>
<tr>
<td>3</td>
<td>Backend</td>
<td>Med</td>
<td>Med</td>
<td>business logic</td>
</tr>
<tr>
<td>4</td>
<td>Backend</td>
<td>Lots</td>
<td>Med</td>
<td>service (memcachd)</td>
</tr>
<tr>
<td>5</td>
<td>Backend</td>
<td>Med</td>
<td>Fast</td>
<td>RT auctions</td>
</tr>
</tbody>
</table>
Cake – having *and* eating II

Embedded + Backend goals have converged...
...but workloads still very different

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>RAM</th>
<th>CPU</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Embedded</td>
<td>Low</td>
<td>Slow</td>
<td>simple RT tasks</td>
</tr>
<tr>
<td>2</td>
<td>Embedded</td>
<td>Med</td>
<td>Med</td>
<td>simple RT + general tasks</td>
</tr>
<tr>
<td>3</td>
<td>Backend</td>
<td>Med</td>
<td>Med</td>
<td>business logic</td>
</tr>
<tr>
<td>4</td>
<td>Backend</td>
<td>Lots</td>
<td>Med</td>
<td>service (memcached)</td>
</tr>
<tr>
<td>5</td>
<td>Backend</td>
<td>Med</td>
<td>Fast</td>
<td>RT auctions</td>
</tr>
</tbody>
</table>

Can a single OS accommodate all workloads and goals?
Cake – having *and* eating III

? requires:

- *Customizable system* – custom resource mgmt
- *Transparent fault tolerance* – auto. reboot system pieces
- *System optimization of parallelism* – map SW → HW
Monolithic Systems

Applications

Web Browser
Web Server
P2P App

Protection Domains

Kernel/OS

File System
Networking
Scheduler
Memory Mgmt

Hardware

Common structuring for desktop/server OSes
Virtual Machines

- Increased isolation – but increased code surface
- Heavyweight
- Custom resource mgmt
  ...to a point
Application-Specific and Fault Tolerant Systems

μ-kernels: e.g. L4, Symbian, QNX, Eros

- Definition???
- Protection domains provide fault isolation
Application-Specific and Fault Tolerant Systems

\( \mu \)-kernels: e.g. L4, Symbian, QNX, Eros

- Definition???
- Protection domains provide fault isolation
- Communication cost limit efficiency
System policies/abstractions are *components*
- user-level
- separate protection domains

Components interact via *invocation*
- function call semantics
- contractually specified interfaces

Low-level functions are components
- scheduling
- memory mapping
- I/O processing

**Composite Webserver**

vs. $\mu$-kernel
- less in the kernel
- *component* definition of all software
  - reusability
  - configurability
Popular component-based framework in common use?
Popular component-based framework in common use?

```
cat foo.txt | grep 'advos' | wc -l
```

How do you write a webserver using command line?

How do you

- control interrupt scheduling
- enforce SLAs
- differentiated service

using the command line?
Can component-based OSes can have performance competitive with existing systems?
Apache, Composite Comparison

![Bar chart comparing Apache, Composite, and FastCGI]

- **Connections/Second (x1000)**
  - **Full Isolation**
  - **No Isolation**

- **Components**:
  - Static File
  - CGI
  - Module
  - FastCGI

- **Patterns**:
  - **Composite**
  - **Apache**
**COMPOSITE Features**

**C^3**: Computational Crash Cart
- system components can fail at any time
- $\mu$-reboot them! – 20$\mu$-sec vs Linux reboot...

**MC^2**: Multi-Core Composite
- control parallelism in each component
- avoid scalability bottlenecks
- use the caches to your advantage!

**TMem**: Transient Memory
- new paradigm for memory management
- predictable, efficient management of memory
- $\rightarrow$ guarantees on how long it takes to get memory!
Wrapping up

1 → N

Customizable
Efficient
Reliable
Scalable
Predictable

? == Composite
Thank You!

composite.seas.gwu.edu

Documentation: github.com/gparmer/Composite/docs/
Code: github.com/gparmer/Composite
Mailing list: compositeos@googlegroups.com