Resource containers:

A new facility for resource management in server systems

Authors:
Gaurav Banga
Peter Druschel
Jeffrey C. Mongul

Presented by: Cassiano Campes

OSDI 1999

The high-performance servers

- Researchers are giving attention to improve web servers performance
- Previous servers used one-process per connection
- New servers use single-process to reduce context-switching costs

* DoS = Denial-of-Service

Quality-of-Service from service providers

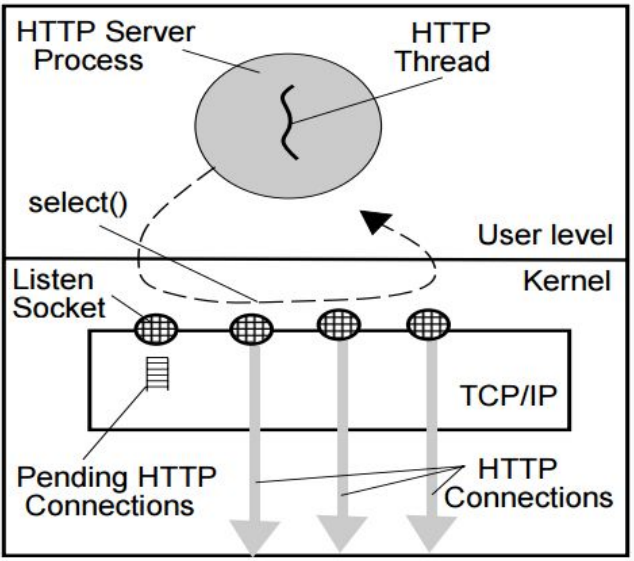
- How to give differentiated QoS to clients
- How control resource usage
  - Application has no control over the consumption of resources inside kernel
- How application can control which connections are high priority

No “connection” between Application and kernel (resource management)

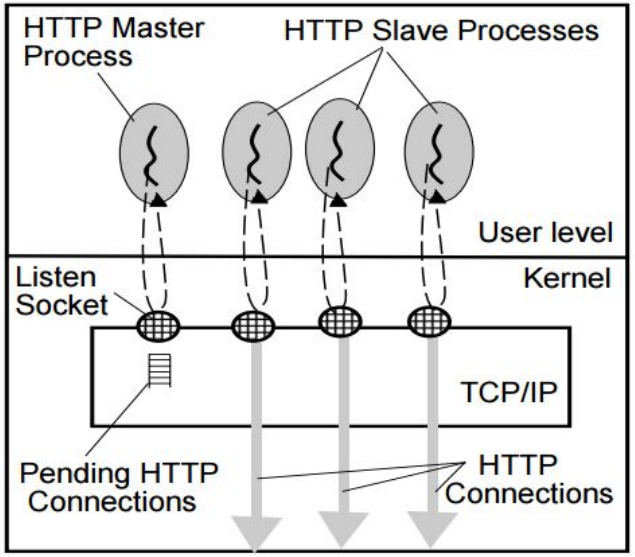
* QoS = Quality-of-Service

Typical models for high-performance servers

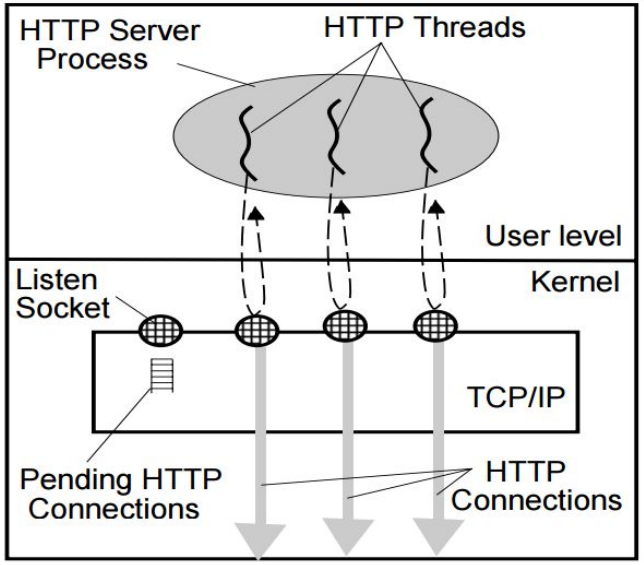
Event-driven server



Process-per connection



Single-process multi-threaded



Uses select() or poll()

Master process attach connection

Thread scheduler is responsible for time sharing

Process Resource Management

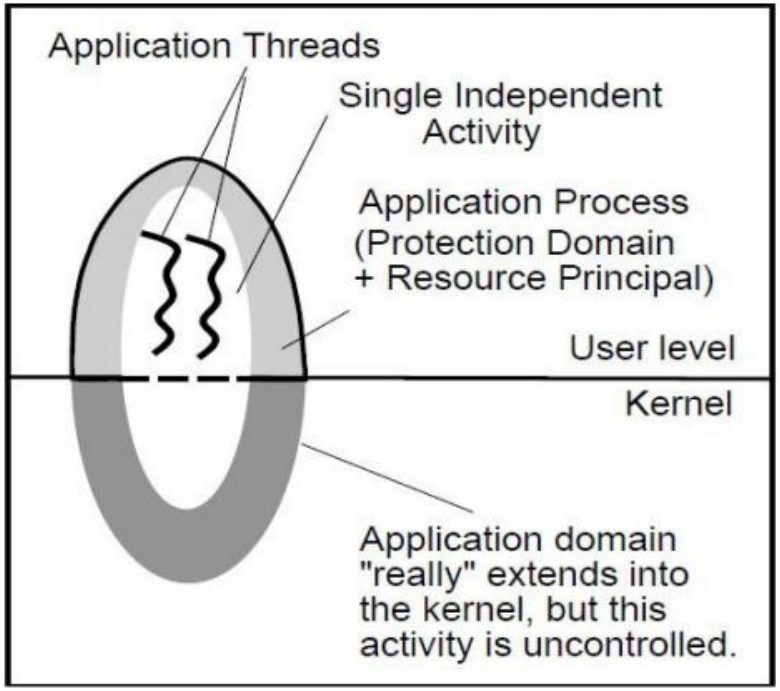
Domain-classification of processes

- Network-intensive application
- Multi-process application
- Single-process multi-threaded application

Network-intensive application

- Multiple threads doing single activity
- Lot of process done in kernel

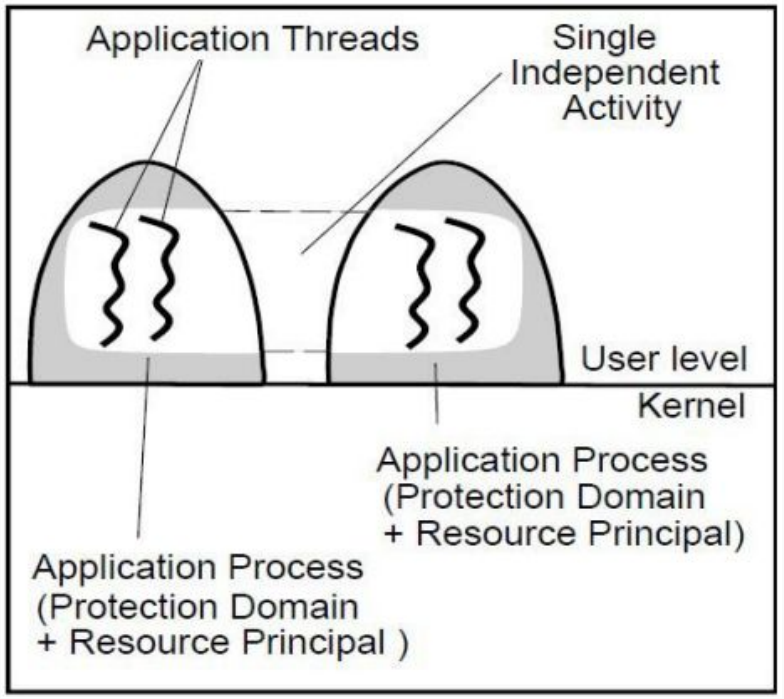
Kernel generally does not control accounting for such resource consumption



Multi-process application

- Multiple user space processes cooperating to perform a single activity
- Managing is done in all process rather than of individual processes

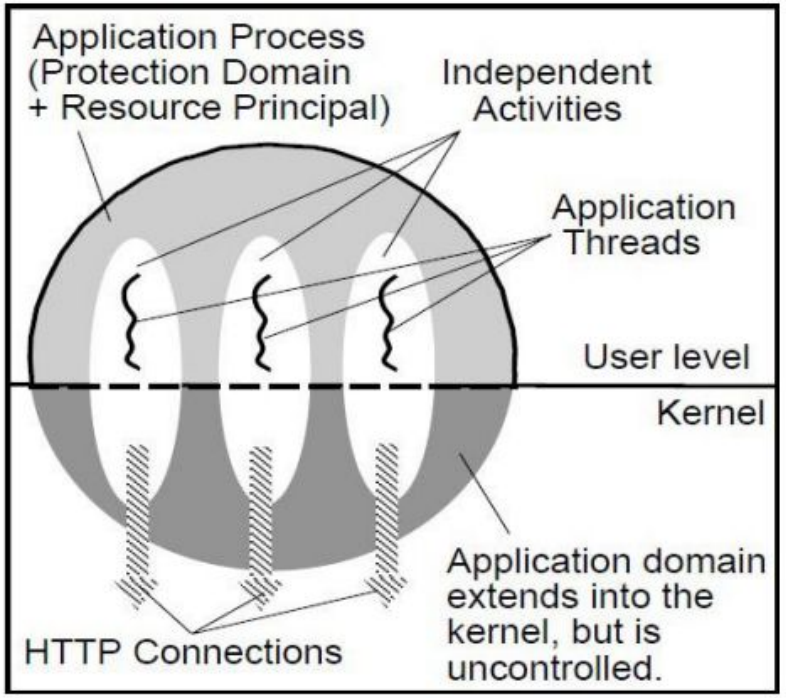
Resource management is a set of all the processes.



Single-process multi-threaded application

- Single process using multiple threads
  - One for each connection

Resource management is a set of all the resources used for the independent activity.



The Motivation Of the Research

Problems

- General-purpose OS provides inadequate support for resource management
- Resource management is tied to processes running to a given machine
- Applications have little control over resources the kernel uses for them
- Resources used by the kernel are often accounted / utilized inaccurately
  - (according to the process) resulting in bad scheduling decisions

Processes (protection domains) are the unit of resource management - the "resource principal"

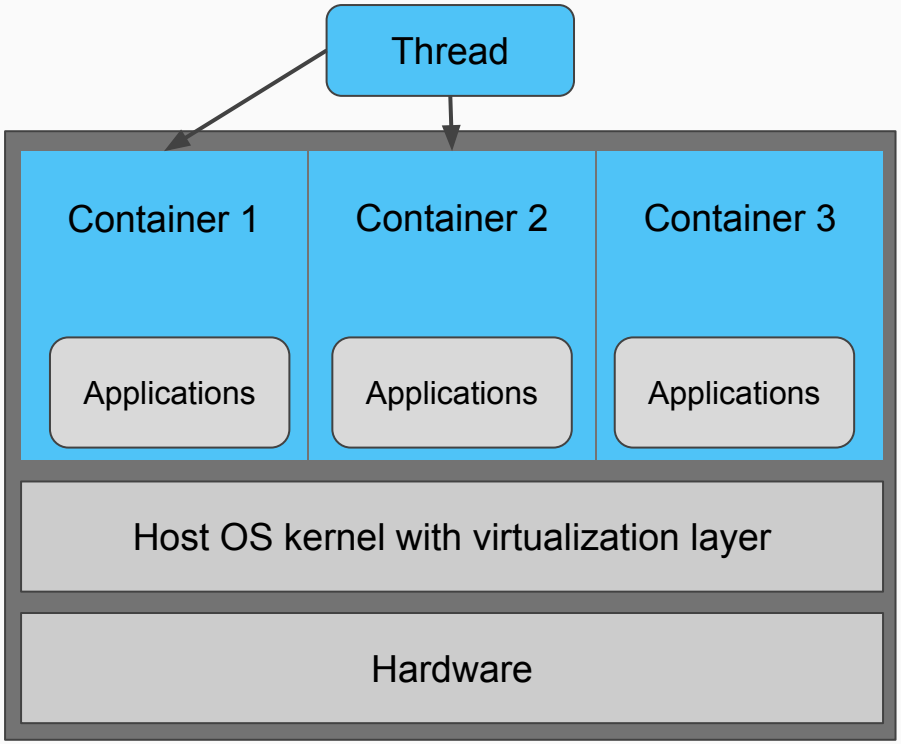
Resource Container

Resource Containers

- Abstraction entity that logically contains all system resources
  - CPU time, sockets, network buffers, etc.
- Containers can also be attached with attributes
  - Limit resources such as: CPU availability, Network QoS, scheduling priorities, etc.
- Resource container aspects
  - Hierarchy (parent -> children)
  - Dynamic resource binding

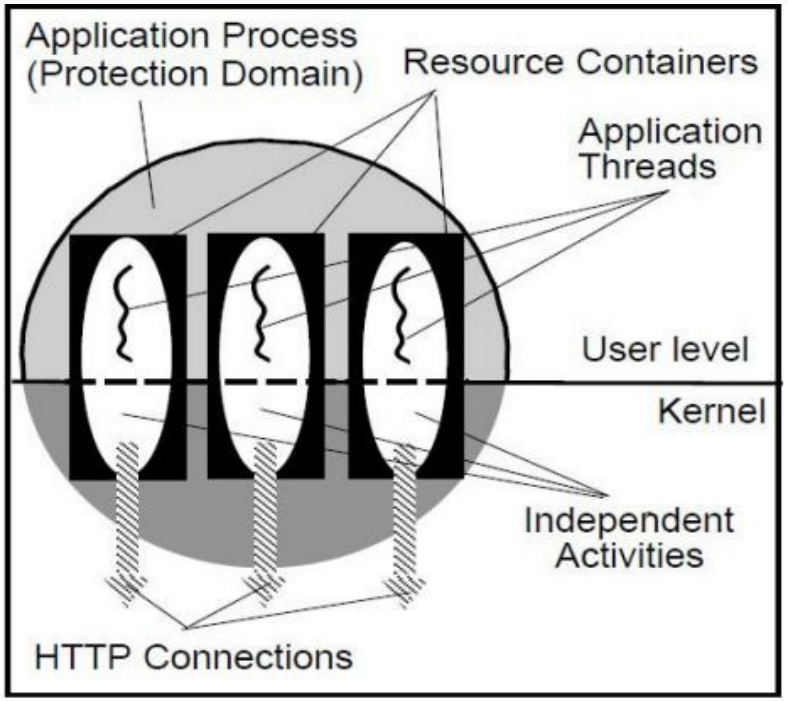
How does it works?

- Applications have to identify resource principals
  - Associate those independent activities with resource containers



Containers in a multi-threaded server

- New RC created for each connection
- Connection uses lot of system resources
  - Usage charged to the corresponding container

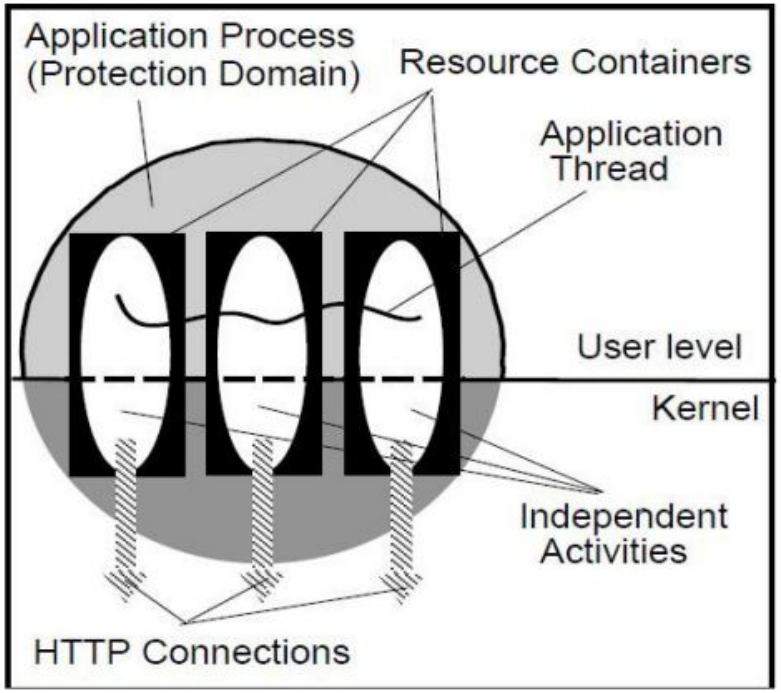


* RC = Resource container

Containers in an event-driven server

- Web Server associates a new container for each connection
  - Single-thread serviced
- Thread's binding changes dynamically as it moves across connections
- The associated container will be charged for the processing the thread performs

To avoid rescheduling threads after every resource container binding, a list of containers is associated with a thread and it is used to schedule the thread



Resource Containers summary

- RC allow an application to associate scheduling info with an activity
  - Allows system's scheduler to provide resources directly to an activity
- Container mechanism supports a large variety of scheduling models
  - Numeric priorities, guaranteed CPU shares, or CPU usage limits
- Scheduler binding between each thread being multiplexed
  - Account the threads that have been shared in multiple containers
  - A thread scheduler binding is set implicitly

LRP: for such a network processing, the kernel does minimal processing and gives the remainder to the application

* LRP = Lazy Receiver Processing

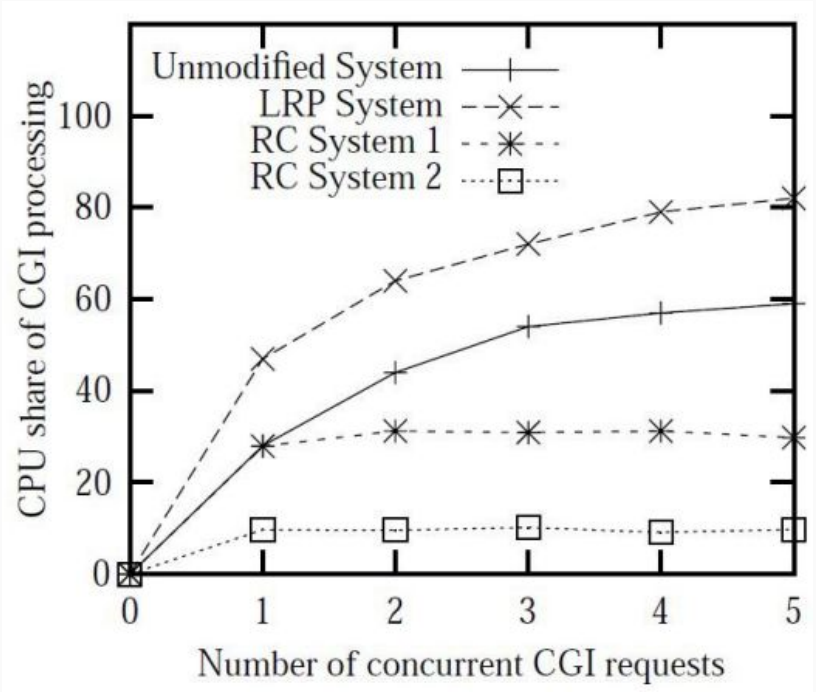
Evaluation

Prototype implementation

- Modifications in Digital UNIX 4.0D kernel
- Changes in CPU scheduler, resource management and network subsystem
- Per-process kernel thread used for processing network packets in priority order of their containers
  - For accounting, this thread sets its resource binding appropriately

Prioritized handling of clients

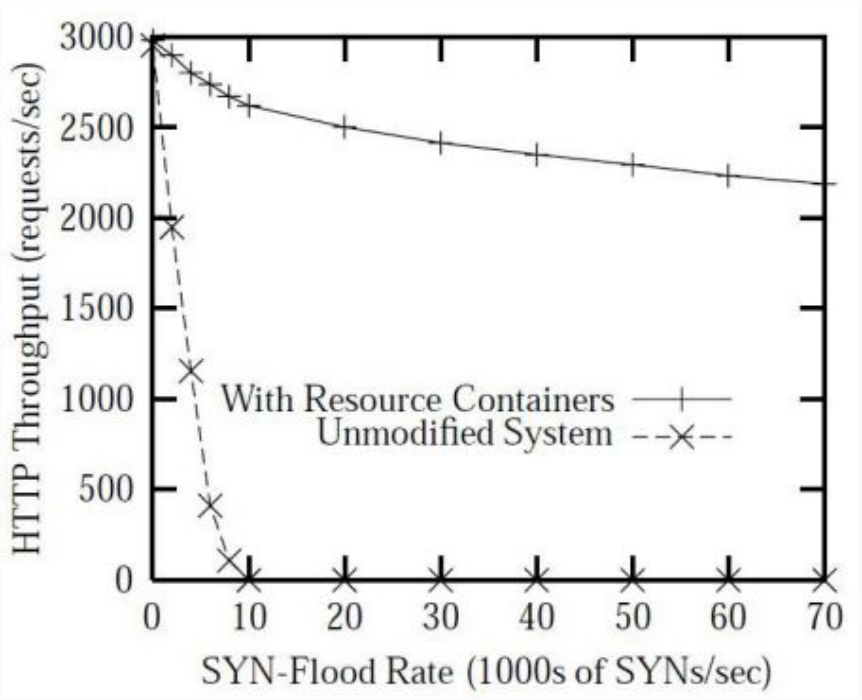
- As the number of CGI increases, CPU is shared among a larger set of processes



* Common Gateway Interface

Immunity against SYN-flood

- Malicious clients sent bogus SYN packets to server at high rate
- Measured server's throughput
  - Slight degradation due to the interrupt overhead from SYN flood
  - Kept above 73% throughput



Conclusion

Conclusion

- Resource container can explicitly identify a resource principal
- Explicit and fine-grained control over resource consumption
  - In all levels of the system (user & kernel space)
- Separation of resource management from protection domain
- Can be used to address a large variety of resource management scenarios

Thank you!

Any Questions?

cassiano.campes@csl.skku.edu

Computer Systems Laboratory