Resource Containers: A New Facility for Resource Management in Server Systems

Gaurav Banga, Peter Druschel, Jeffrey C. Mogul

February 1999

Presented for EECS 443, Northwestern University
Stephen Brenner
January 26, 2009

Overview

- Introduction
- Traditional Servers
- Resource Containers
- Benefits of Resource Containers
- Summary

General Implementation

- Processes are used for protection domains and as resource primitives.
- Easy and straightforward, but can be inefficient when this model does not match ideally, as in the case of servers.
  - Examples will an HTTP server, although this could apply to other servers, as well.

Traditional Servers

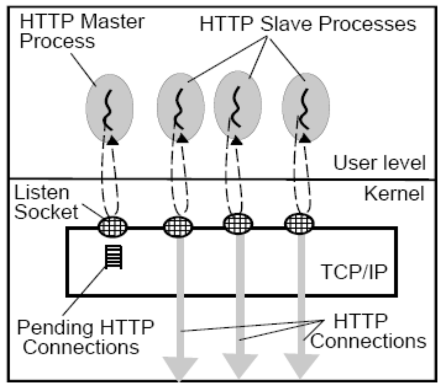


Fig. 1: A process-per connection HTTP server with a master process.

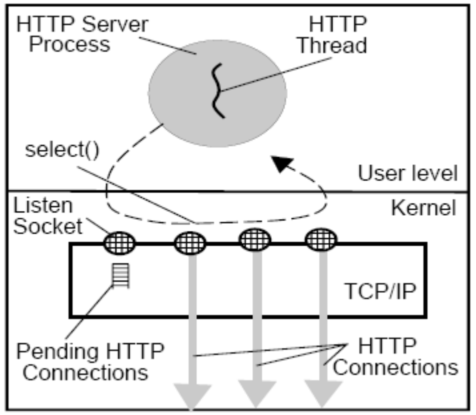


Fig. 2: A single-process event-driven server.

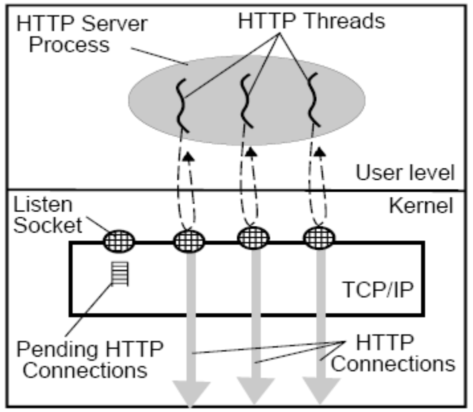


Fig. 3: A single-process multi-threaded server.

- Traditional servers have multiple models.
  - Above, one master process gives work to multiple pre-forked working slave processes.
  - Right above, a single process is used to eliminate context switching and IPC costs.
  - Right below, a multithreaded process uses one thread per connection.

Downsides of Traditional Servers

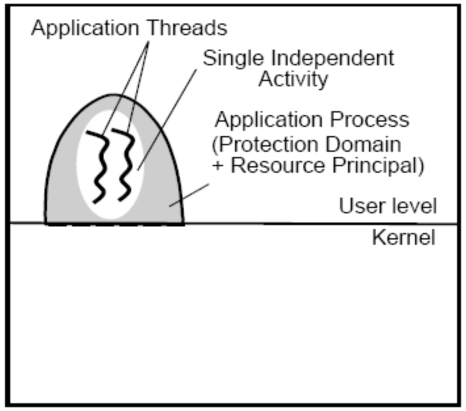

Application Threads
Single Independent Activity
Application Process (Protection Domain + Resource Principal)
User level
Kernel

Fig. 4: A classical application.

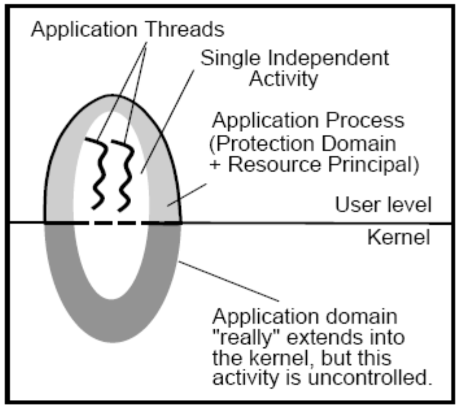

Application Threads
Single Independent Activity
Application Process (Protection Domain + Resource Principal)
User level
Kernel
Application domain "really" extends into the kernel, but this activity is uncontrolled.

Fig. 5: A classical network-intensive application.

- Traditional models do not work well when the protection domain should not match the resource principal.
  - Above, the traditional model works well for an application that does not go into the kernel much.
  - Right above, the traditional model fails to associate a network-intensive application's work in the kernel with the application.
  - Right below, a multiple process application performing a single independent task cannot share the same resource principal in this model.

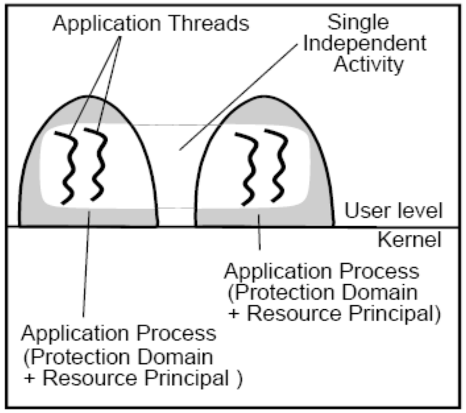

Application Threads
Single Independent Activity
User level
Kernel
Application Process (Protection Domain + Resource Principal)
Application Process (Protection Domain + Resource Principal)

Fig. 6: A multi-process application.

Downsides of Traditional Servers

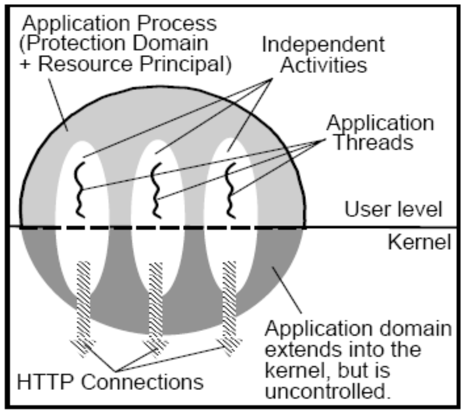


Fig. 7: A single-process multi-threaded server.

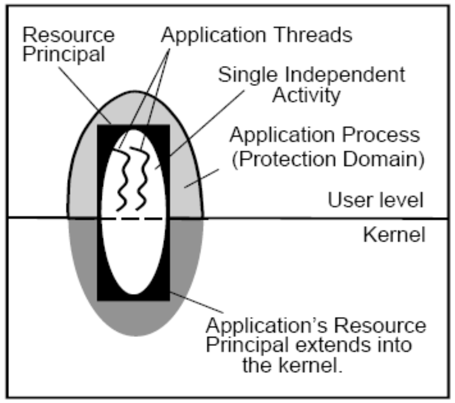


Fig. 8: A network-intensive application in a LRP system.

- Traditional models do not work well when the protection domain should not match the resource principal.
  - Left, a multithreaded application performing different independent activities is required to share the same resource principals.
  - Right, a modified kernel using Lazy Receiver Processing (LRP) associates the work done in the kernel with the correct application. This is a more accurate model, but it still keeps the protection domain and the resource principal the same.

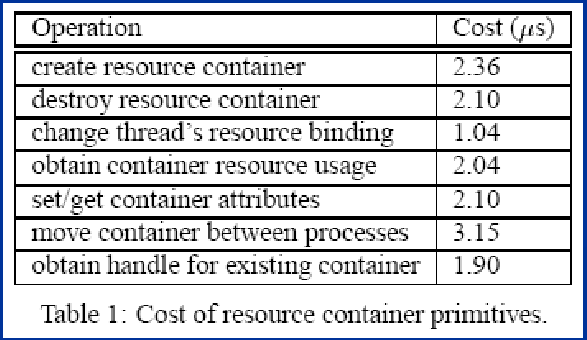
Resource Containers

- An entity that holds all system resources of an application for an independent activity used for scheduling.
  - Example: a Web server would include connection CPU time, sockets and other kernel objects, protocol control blocks, and network buffers used.
- Containers have attributes for scheduling, resource limits, and network QoS values.
- Can be hierarchical.
- Used in user and kernel modes.

Operations on Resource Containers

- Creating a new container
- Set a container's parent
- Container release
- Sharing containers between processes
- Container attributes
- Container usage information

- Operations for relationship control between containers, threads, sockets, and files
  - Binding a thread to a container
  - Reset the scheduler binding
  - Binding a socket or file to a container



Operation	Cost (μs)
create resource container	2.36
destroy resource container	2.10
change thread's resource binding	1.04
obtain container resource usage	2.04
set/get container attributes	2.10
move container between processes	3.15
obtain handle for existing container	1.90

Table 1: Cost of resource container primitives.

- Extra costs of resource containers are relatively insignificant

Servers Using Resource Containers

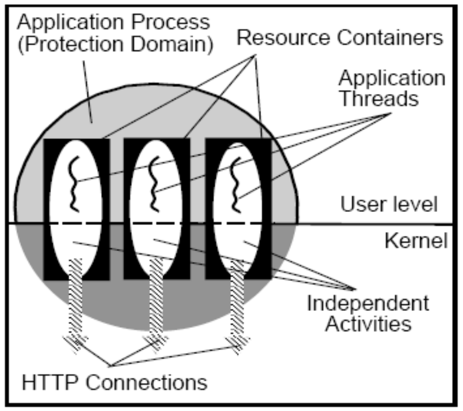


Fig. 9: Containers in a multi-threaded server.

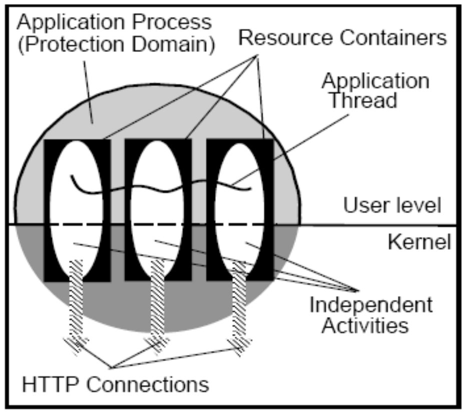


Fig. 10: Containers in an event-driven server.

- A new resource container is created for each new connection.
- That resource container is charged with the processing, including kernel processing.

Performance Benefits

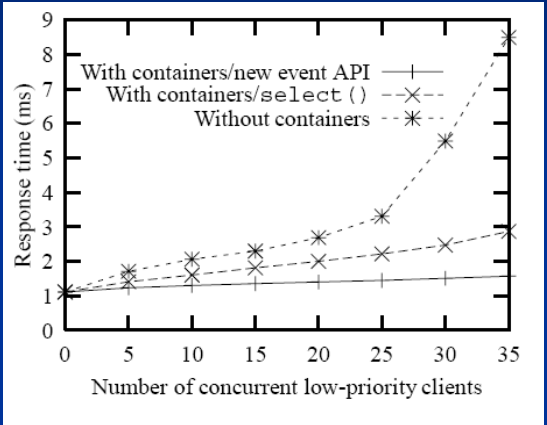


Fig. 11: How T_{high} varies with load.

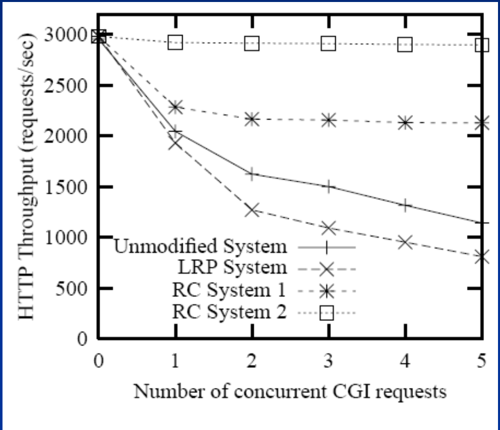


Fig. 12: Throughput with competing CGI requests.

- Using resource containers can give high priority tasks or connections more processing power.
- Graph above shows response time for a single high-priority client with an increasing number of low-priority clients connecting.
- Graphs to the right show throughput (above) and CPU usage of CGI (below) using a 1 KB cached static document and increasing amounts of CGI requests.

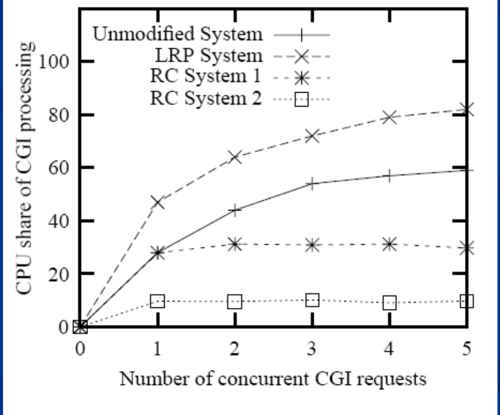


Fig. 13: CPU share of CGI requests.

Protection Benefits

- Using connection filters, malicious connections can be placed into a resource container where it can be starved to protect the system.

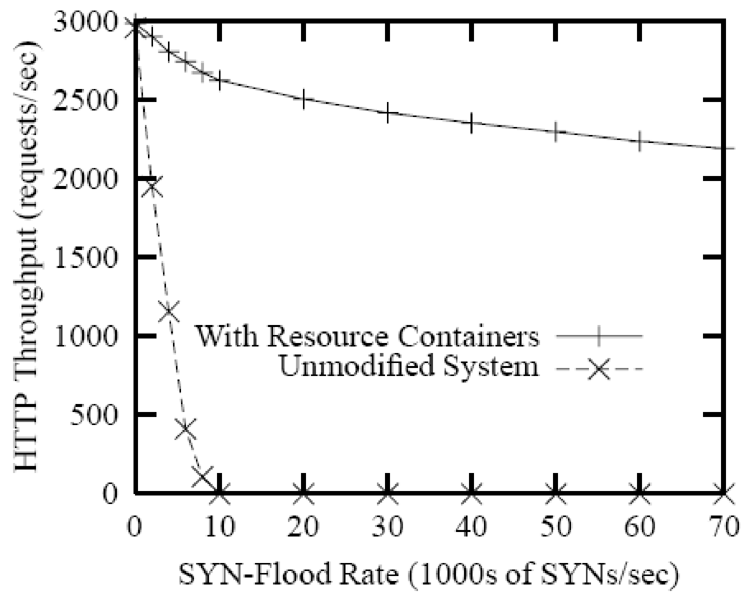


Fig. 14: Server behavior under SYN-flooding attack.

Virtual Server Isolation

- Running multiple virtual servers on the same machine, each in its own resource container, an administrator can limit the resources each gets.
- Resources can then be divided proportionately to how much is being charged for each to run.

Summary

- By breaking the traditional model that a protection domain and a resource principal must coincide, many benefits are derived.

Any questions?

Thanks