The GNU Hurd: Towards Extensibility

Neal H. Walfield

neal@gnu.org

FOSDEM, 2005

Hurd Design Goals[1]

- Increased Flexibility
  - Translators
  - Modifiable VFS
  - Replacement of system components
- Increased Security
  - Fault isolation
  - User-space servers
  - Authorization tokens, not DNA
    - Fine grained control
    - Limit lateral access (unintended sharing)

[1] Michael Bushnell, Towards a New Strategy of OS Design

Have We Succeeded?

- Yes!

Well, Mach has Important Limitations

- Inadequate Memory Object API
  - All users of a memory object get the same access rights: if one user has write access, all users get write access
- Resource Accounting
  - Servers allocate resources on behalf of clients
    - No resource limits
    - No quality of service
- Performance and State of GNU Mach
  - Optimized for early 1990s hardware
  - Page out daemon evicts pages one at a time
  - Driver code is outdated and not well integrated

Can We Fix These Issues?

- Inadequate Memory Object API
  - Non-invasive API improvements
  - Additional level of indirection
- Resource Accounting
  - API assumes that servers allocate resources on behalf of clients
  - Deep, impractical API changes required
- Performance and State of GNU Mach
  - Can tune algorithms and clean up code

Are These Changes Enough?

- Inadequate Memory Object API
  - Fixed
- Resource Accounting
  - Many systems make due without: just add, e.g. an out of memory killer
- Performance and State of the Code
  - Resource scheduling algorithms are designed for a monolithic system and perform poorly in a multi-server environment
  - Dramatic improvement requires a shift in the framework

Global Resource Scheduling

- In 1968, Denning states that the OS cannot consult applications for scheduling advice if it is to be fair; the OS must rely only on observed behavior [2]
- All Popular Operating Systems still following this philosophy
- Approach is not straightforward: Linux is constantly being tuned for the *typical* work load
  - Lots of VM tuning patches
  - Lots of CPU schedulers (Kolivas' Staircase, Williams' Zaphod Priority)

[2] The Working Set Model for Program Behavior

Problems with Global Resource Scheduling

- Global view is insufficient
  - Good resource scheduling requires application specific knowledge
  - Ever hear an audio playback “skip” when a program started doing a lot of disk I/O?
  - Problem: lack of quality of service
  - Stonebraker notes that applications can know page usage while the kernel can only ever guess based on a predetermined set of access patterns which it can recognize[3]
  - Bad for
    - Databases
    - Garbage Collectors
    - Scientific Applications
    - Multimedia Applications (best effort)

[3] Operating System Support for Database Management

But Linux Performs Well

- Linux has local knowledge of how many resource intense applications work
  - Drivers
  - File Systems
  - Network Stacks
- We can only do worse on a multiserver architecture. As Linus Torvalds recently said:
  “I really do believe that user-space filesystems have problems. There’s a reason we tend to do them in kernel space . . . Guys, there is a reason why microkernels suck. This is an example of how things are not ‘independent’. The filesystems depend on the VM, and the VM depends on the filesystem. You can’t just split them up as if they were two separate things (or rather: you can split them up, but they still very much need to know about each other in very intimate ways)[4]”

[4] 18 Nov 2004 email to LKML

Listening to Linus

- Two Ways to keep VM and FS intimate
  - Move file systems back into the kernel
  - Move VM into user space
- Former prevents flexibility
- Latter supports Hurd philosophy

Extending Mach

- Premo Pagers[5]
  - Extension to Memory Objects
  - Memory Object's paging policy in server
  - Inflexible: when Mach a server to evict a page it must be from a particular memory object
  - Step closer to application but in the application
- High Performance External Virtual Memory Caching Mechanism (HIPEC)[6]
  - Extension to Memory Object
  - Upload trusted code to Mach
  - Code unable to access application state

[5] McNamee, Extending The Mach External Pager Interface To Accommodate User-Level Page Replacement

[6] Lee, In-Kernel Policy Interpretation for Application-Specific Memory Caching Management

Other Systems

- V++[7]
  - Application Kernels
  - Market Model for allocating sparse resources
- Aegis (Exokernel)[8]
  - Exports everything to user-space, even page tables
  - Library OS
  - No quality of service guarantees
- Nemesis[9]
  - Self-paging tasks
  - Guaranteed pages on medium term contracts
  - Specifically address quality of service

---

[7] Harty, Application-Controlled Physical Memory using External Page-Cache Management

[8] Engler: AVM-Level Virtual Memory

[9] Hand, Self-Paging in the Nemesis Operating System

Hurd on L4 Approach

- Self-paging tasks with no fall-back (default) pager
- Late Virtualization
  - Applications are allocated physical resources; their responsibility to multiplex them (if needed)
- Extensible libraries implement widely used policies

Resource Allocation Revisited

- Applications allocate resources
- Applications pass resources to servers
- Servers do minimal allocation on clients' behalves (only metadata, e.g. connection state)

Degree of Extensibility

- Other important scarce resources: CPU and I/O
  - Applications can scale their use according to availability and provide different service
  - Denning noted that memory scheduling is tightly coupled to CPU scheduling: if a task has memory but no CPU time, it can't make progress and the memory is wasted[10]
- Can we apply the same technique to these?

[10] The Working Set Model for Program Behavior

Separating Control and Data Paths

- Filesystems perform two tasks: structure bits and fetch data
- If clients can fetch data directly from backing store then the filesystem manages metadata and clients can talk directly to back store and negotiate quality of service parameters
- Algorithm
  - Client wants to read file
  - Filesystem determines block list
  - Tells backing store to allow client to access block list
  - Filesystem returns block list and server to client
  - Client contacts the device driver and asks it to read the block list

Sounds Great, But. . .

- How do we allow efficient, secure access to the disk?
- If we find a way to do this efficiently and our physical resource scheduler is efficient and fair, applications will optimize for their best interests (or hurt themselves) then the system will perform more efficiently

Where Are We Now?

- Physmem framework is in place
- Proof of concept
- Need to develop the global allocation policy
- Need to implement the default paging policy library
- Good feeling. . .

Big Questions

- What policy should we use to allocate physical resources? A market based model like V++? Needs more research.
- Can (and how do) we implement a user space disk efficiently?

Copyright 2005 Neal H. Walfield

This work is licensed under: Creative Commons Attribution-ShareAlike 2.0 license. You are free:

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

- Attribution: You must give the original author credit.
- Share Alike: If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

For any reuse or distribution, you must make clear to others the license terms of this work. Any of these conditions can be waived if you get permission from the copyright holder.
Your fair use and other rights are in no way affected by the above.
Full text at
http://creativecommons.org/licenses/by-sa/2.0/legalcode .