CSE221 Lecture 8

Aronya Baksy

October 2024

1 Plan 9

1.0.1 Goals

- Build centralized, heterogeneous system consisting of cheap PCs
- Cheap microcomputers, expensive central computers/file servers made up from shared memory multiprocessors
- Adapt good ideas from UNIX and add some new ones: use FS to coordinate access to all resources (including h/w)
- Innovations in file systems: network access protocols, per-process file systems and namespaces
- Address issues with UNIX and bring in new tools (compilers/languages/libs/window systems)

1.1 Design Principles

- All resources named and accessed like files
- Standard protocol for accessing these resources
- Disjoint hierarchies provided by each service are joined in a single namespace
- The client's local name space provides a way to customize the user's view of the network
- user namespace customizes view of resources no global view for users.
- services that expose file hierarchies: I/O devices, backup services, the window system, network interfaces, etc. (e.g. the `/proc` filesystem that provides clean interfaces to inspect processes)

1.2 File System Design

1.2.1 9P Protocol

- network file access protocol that is used for all commmunication
- ties into the file abstraction for all resources
- single unified protocol instead of different protocols for different communication types
- central file server stores permanent files and presents them to the network as a file hierarchy (using 9P)
- Storage is hierarchically arranged as memory, local disk and WORM server (each is a subset of the next level)
- WORM drive stores backups of the local disk at 5pm every day using a block-queueing mechanism
- The window system called 8.5 offers a file-system interface which client programs can write to for displaying text
- 8.5 is a file server that multiplexes the files in `/dev` to all the clients
- FTP is mounted on `/n/ftp` and can be used as a normal file system by the client after mount
- exportfs is a user process that takes a portion of its own name space and makes it available to other processes
- import command mounts remote file systems on the client, while `cpu` command starts a remote process and shares the /dev directory with the remote server

1.3 Configuration and Administration

- All system files reside only one one main file server (additional file servers only add storage) which makes admin easy
- Highly portable, and does not require expensive hardware upgrades frequently

1.4 Programming Interfaces

- Uses mostly ANSI C but also introduces a new dialect for parallel programming called Alef (uses UTF-8 encoding, removes some macros like #if and #ifdef)
- Alef adds synchronization primitives like channels, queuing locks, etc.
- THe `rfork` system call creates new processes:
  - Takes in a bit vector which denotes which parent resources will be shared, copied or created in the child process
  - Threads can be created by sharing all resources b/w parent and child, used by Alef
  - rfork can be used to modify own resources too (instead of creating new child)
- spinlocks are provided by h/w dependent libraries at user level

1.5 Namespaces

- Private local namespaces exist per user, can be viewed on any of the terminals in the Plan 9 installation
- All filenames are relative to a namespace, but local names are enfoced by a global convention (e.g. /dev, /proc to store I/O device and process information)
- These namespaces can be customized, mounted or unmounted
- Plan 9 also offers union directories where several real directories can be mounted at the same point (the union of two directories is simply the concatenation of their contents)

2 LegoOS

2.1 Goals

- **Splitkernel** model: OS split into loosely coupled monitors that run on, manage allocation and handle failure of specific h/w components
- propose a new hardware architecture to cleanly separate processor and memory hardware functionalities
- **H/w disaggregation**: all hardware organized as independent, failure-isolated, network-attached components with their own controllers

2.2 Splitkernel

- Individual kernels communicate only through message passing (no explicit coherence)
- Three types of monitors: process, memory, storage monitors
- Hardware support for memory mgmt (because it is removed from process mgmt):
  - all processor caches organized as virtua memory
  - additional last-level cache uses a part of the DRAM
  - 2-level virt mem management, memory replication for DR
- Each hardware component has a controller that can run the monitor for that component (e.g.: FPGA or ASIC)
- No guarantee of data coherence across components, applications implement it through message passing
- Global resource management for coarse-grained decision making upon failure but otherwise splitkernel-level decisions

2.3 Disaggregation of Hardware Resources

2.3.1 Disadvantage of Monolithic Servers

- Inefficient resource utilization
- Poor h/w elasticity (tough to upgrade)
- Coarse-grained failure domain
- Does not support heterogeneous components

2.3.2 Resource Disaggregation

- Trends: fast increasing network bandwidths and speeds, hardware-network interfaces without need for software, more processing power in hardware
- Running traditional monolithic kernels and accessing remote memory/storage adds network latency, doesn't take adv. of hardware, makes the processor a SPOF

2.4 LegoOS Design

- Main abstraction exposed to user: vNode or virtual node (each vNode has an ID, a virtual IP addr, a mount point, process isolation and security)
- No shared memory across processors but threads inside the same process share memory (apps using shared memory need to be re-implemented to use msg passing)
- pComp, mComp and sComp are independent h/w resources each having controller and n/w interface
- pComps handle only caches. all other mem like paging, TLB etc. is on mComps
- pComps only see virtual memory addresses for their caches. all caches are virtually indexed and tagged. problems with virt caches:
  - synonym, i.e. multiple virtual addr map to a single physical address, since LegoOS does not allow writable memory sharing this is solved
  - homonym, i.e. multiple processors using the same virt addr for diff phy locations, this is solved by specifying ASID in each cache line
- Exploit locality (space/time) of memory access to reduce possible network latencies when doing mem access
- Each pComp's DRAM is organized as a cache under the LLC, called *ExCache*. virtual, inclusive cache, handled only by h/w
- ExCache miss is handled in software by the process monitor (supports FIFO or LRU eviction)
- Compat layer above the process monitor at each pComp to store Linux state for full ABI compatibility

2.4.1 Scheduling

- Goal: optimize number of context switches and scheduling overhead instead of CPU utilization
- No kernel pre-emption or interrupt mechanisms because I/o is very fast (threads use busy waiting for I/o requests)

2.4.2 Memory Management

- memory monitor manages both the virtual and physical memory address spaces, their allocation, deallocation, and memory address mapping
- process address space span multiple mComps, one *home mComp* to intially load the process and handle all memory related requests/syscalls
- A vRegion is a portion of the virtual address space that is owned by a single m-Component, is around 1GB in size. One p-comp uses multiple vRegions
- The vmaTree (virtual memory area) is maintained by the owner of a vRegion, contains info about address range and protection

2.5 Evaluation

- The actual implementation uses commodity hardware and p, m comps are simulated in software. This adds overhead while doing benchmarks
- LegoOS network stacks (socket over InfiniBand and RPC over InfiniBand) outperform native Linux
- using more worker threads per mComponent and using more mComponents both improve throughput when an application has high parallelism, but only till 4 worker threads
- LegoOS performs worse with higher workload threads, because of m-Comp bottlenecks
- Only 1.6-1.3x slowdown for real world apps (LegoOS excache size is 25% of the working set size) compared to Linux server having sufficient memory to fit an entire working set