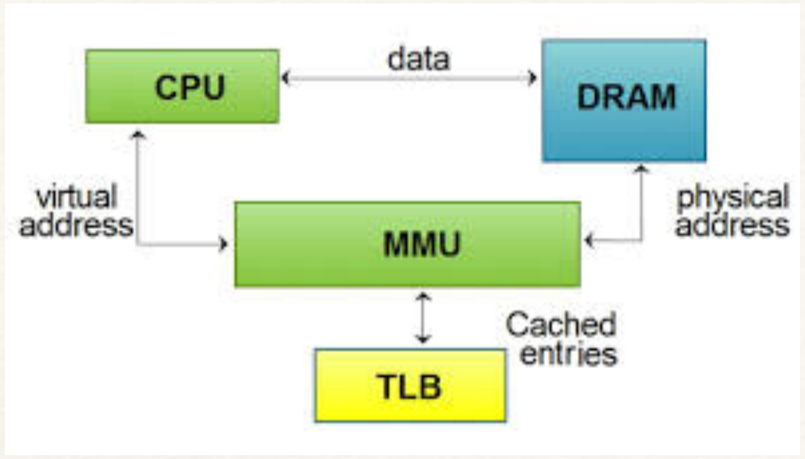


HELP WANTED

Christopher Lameter <cl@linux.com>

Fundamental Trends and Issues in Memory Management

Linux Forum Korea 2015

Introduction

- MM provides On demand Paging, Segmentation and Virtual Memory
- Memory Management is central to an operating system running applications in a controlled way with decent performance.
- Memory management is fundamental to Modern computing
- Isolation / Virtualization
- Changes in hardware and application features drive memory management evolution.

Linux MM History

- Initial development of MM by Linus to exploit the demand paging of Intel 386 Cpus using 4K pages (early 90s). Reimplementation of the classic Unix VM.
- Large Page support. Huge Pages (2 Megabytes, 2003) and Giant Pages(1GB, 2007)
- NUMA 2003-2005
- Scalable VM statistics (2005)
- Page Migration 2005-2006
- Defragmentation heuristics 2009-?
- Virtualization and resource control (KVM etc, Cgroups, containers) (2007-)
- Heterogeneous Memory (2015).
- Coherent Device Memory(?)

Main Technology Trends

- More memory
- More complex code
- More code
- More processor cache levels
- More work to optimize code to run fast
- Limits on execution speed of a single thread.
- Even more complex memory hierachy through access to device memory with varying performance (NVRAM, GPU, Coprocessors)

Trends in MM

- More difficulties with fragmentation
- Complexity of allocators increases
- The amount of memory allocators increases
- Less and less people want to use swap (why is this curse still following us?)
- More demands for performance on the memory allocators (which in turn leads to the creation of new ones).
- Realtime support will be required soon.

More complex locks

As system size increases

Fairness
Fastness
Locks are evil
Alternates to locking (percpu etc)
Tolerating races
Fuzziness
Complex memory hierarchies create issues for locking.

Everything is Memory and can be used as memory

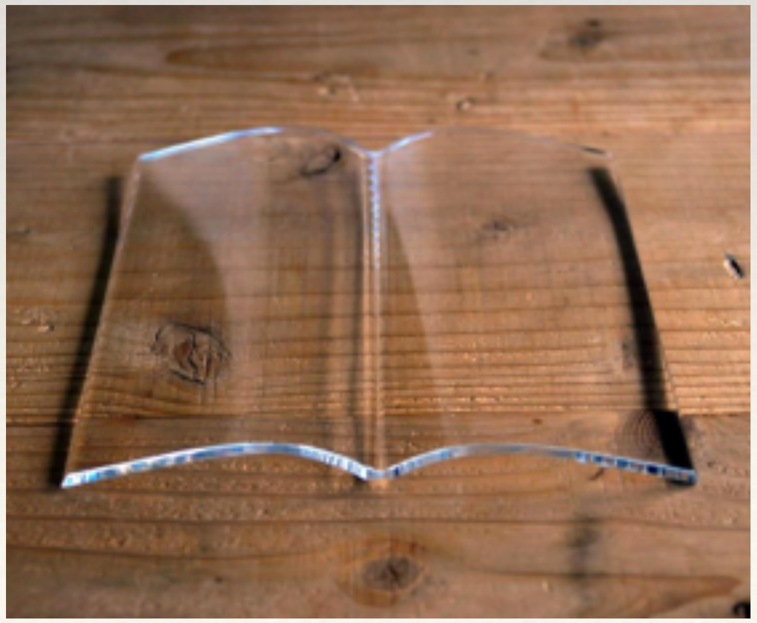
- Devices and their APIs vanish into memory regions
- Flash Memory -> NVRAM
- Storage -> Flash -> NVRAM
- Networking -> RDMA -> Shared memory operations
- Graphics -> Memory mapped data structures.
- What is left? Legacy APIs. Convenience.

Page sizes

- 4k pages and their management are overwhelming the kernel. 4G RAM = 1 mio page structs. 512G systems are possible. Hundreds of millions page structs? Kernel does LRU processing etc etc with these.
- Huge page support needed or support without page structs (DAX). But those are limited in what they support.
- Larger page size reduce metadata overhead and enhance subsystem performance. But defragmentation is an issue that we cannot deal with properly.

Transparent Huge Pages

- Solution to the 4K TLB (PTE) resource issues as well as scaling.
- Uses PMDs in hugetlb mode to refer to pages.
- THP dynamically converts 512 4K pages to one 2M page
- References can exist to the huge page as a PMD as well as PTEs (4K). Complex locking. Complex reference counting.
- Struggling with how to make it scale.
- Using 2M while keeping the illusion of 4K page size alive (for legacy reasons this is important).



Fragmentation or the curse of the 4K page.

Need more contiguous memory

Numerous attempts have failed here.

The fragmentation problem must be solved

Kernel Objects need to be movable!!!!

Various languages have done this in an automated way. Why should the kernel not be able to do this? We did it with the 4K pages and made them movable. Lets make other kernel objects also movable.

Feedback wanted!

Conclusion

Help wanted to deal with issues in MM.