Controlling Memory Footprint at All Layers: Linux Kernel, Applications, Libraries, and Toolchain

Xi Wang
Broadcom Corporation

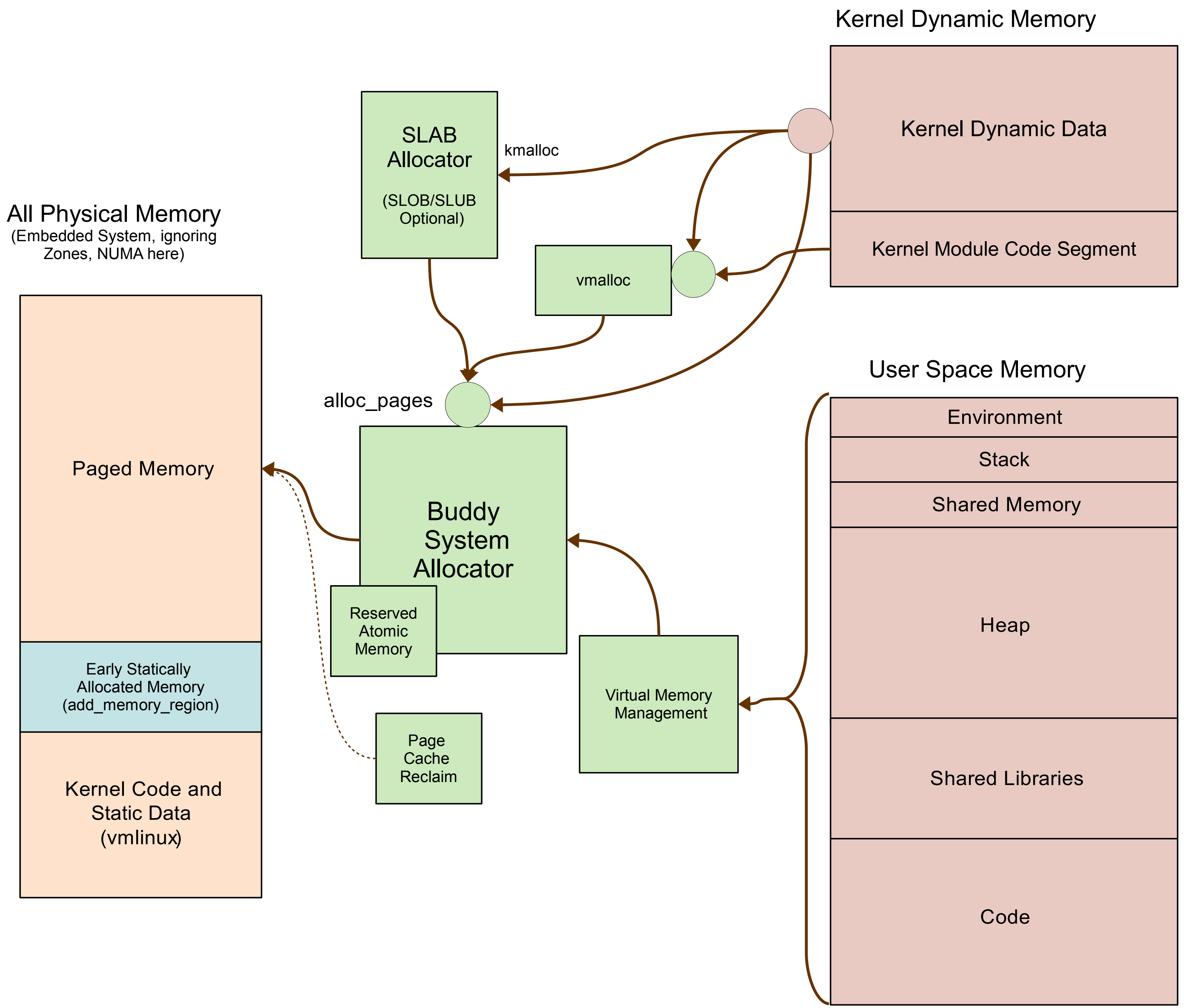
Questions, Comments:
xiwang@broadcom.com
peknap@yahoo.com

Introduction

- **Search with more breadth and depth**
  - Visit more layers
    - This talk is not about vmlinux or kmalloc
  - Not just reducing size. Goal is to add features without adding memory
    - E.g. improve system performance under low memory situation

- **About medium-sized embedded systems**
  - With DRAM + Flash, no swap partition
  - Lightly configured kernel image fits in comfortably, but need more space for everything else
  - Based on 2.6.2x, let me know if there are new changes

Introduction



Kernel Memory Lifecycle

- **Understanding kernel memory lifecycle**
  - Memory is an active subsystem with its own behaviors
  - Kernel gives away memory carelessly and let everyone keep it
    - E.g., file caching, free memory in slab not returned immediately
  - Sometimes try to get it back
    - Periodic reclaiming from kswapd
    - When there is not enough memory
- **Most apparent effect of memory life cycle: Free memory in /proc/meminfo is usually low unless you have a lot of memory**
  - Never use the free memory number in /proc/meminfo as free memory is a fuzzy concept in Linux kernel. Think in terms of memory pressure instead

Improve System Low-Memory Behavior

Low memory system behavior can be poor in embedded systems

- **Symptoms**
  - System slows down significantly when memory is low, but there is still enough memory
  - OOM kills processes too eagerly
- **Reasons**
  - Target system for typical kernel developers is DRAM + hard drive, so some behaviors may not be the test fit for DRAM + Flash embedded systems

Improve System Low-Memory Behavior

- When memory is low, kernel spends a lot of time in kswapd trying swap out pages, but only with limited success – dirty pages have nowhere to go in a swapless system
    - Performance impact
    - More chances for memory-introduced deadlock
- Incremental search for free-able pages would not be optimal for better speed or reducing fragmentation under low memory conditions

```
mmzone.h
 /*
  * The "priority" of VM scanning is how much of the queues we will scan in one
  * go. A value of 12 for DEF_PRIORITY implies that we will scan 1/4096th of the
  * queues ("queue_length >> 12") during an aging round.
  */
  #define DEF_PRIORITY 12
```

Improve System Low-Memory Behavior

- **Keep system running when memory is low**
  - Lower the priority kswap kernel thread
    - Rationale: In a swapless system, it cannot really swap
      - Made a big difference, no serious side effects
  - When trying to free pages, look at more pages at the first try (make DEF_PRIORITY smaller)
    - Not enough data to tell the effect
  - Turn off OOM killer
    - For small closed embedded system, every process is essential, as simple solution is to reboot the system instead of killing processes
- **Excessive page frame reclaim activity can cause other problems as well**
  - Power management
  - Real-time latency

Tap Into Reserves

- **When kernel thinks memory is low?**
- **Parameters of interest: Minimum free memory standards maintained by kernel**
  - Reserved memory pool for GFP_ATOMIC
    - /proc/sys/vm/min_free_kbytes
    - Default value is calculated based on memory size
  - Zone watermarks put a threshold on fragmentation
    - Certain number of free pages at certain size
    - See zone_watermark_ok function for details
- **When the minimum standard cannot be met, kswapd will keep running**
  - Until it meets the standard or calls OOM killer

Tap Into Reserves

- **Tweak possible**
  - Reduce reserved memory pool
    - From sysfs in newer kernels, /proc/sys/vm/min_free_kbytes
  - Lower fragmentation threshold
    - Can change algorithm in zone_watermark_ok function
- **Trade-offs**
  - Running out of memory in interrupt context
  - Not enough big free memory blocks
  - But for embedded systems we have better control, can be running closer to the limit

Fight Fragmentation

- **Fragmented free memory can be as bad as no memory – don't ignore fragmentation**
- **Try not to create fragmentation**
  - Preallocate memory when possible
  - Write a task-specific memory allocator if needed
- **Able to use fragmented memory**
  - Use vmalloc for allocating bigger memory blocks
  - Move code to user space
- **Already discussed**
  - Lower fragmentation threshold
  - When trying to free pages, look at more pages at the first try
    - Not enough data to tell the effect

More on Kernel Memory

- **Alternative allocators**
  - SLOB
  - SLUB
- **Reference**
  - “Understanding the Linux Kernel” Bovet & Cesati. O'Reilly Media

System Design Issue: MMU Is An Advantage, Use It

uClinux may look like an interesting option, but regular kernel with virtual memory is a better bet if you have many user applications

- **The default behavior, read-only "swapping" between Flash and DRAM is beneficial**
  - More programs with less DRAM – load from Flash into DRAM when needed
  - Slows down when overloaded, but graceful degradation
  - Only possible with virtual memory
- **Execute in place may not be a good trade-off**
  - A lot more flash space for a little less DRAM space

Problems with Shared Library

Shifting focus into user space

- **Lower code density of .so**
  - Function need to be ABI-compliant
  - Position independent code
  - Functions cannot be inlined
  - Limit inter-function compiler optimizations
- **Need per-instance memory pages for dynamic linking**
  - Dynamic linking tables (GOT, PLT)
  - Global variables
  - Need to watch this overhead: If 10 processes in the system and each of them is linked to 10 shared libraries, ~400k of free memory is consumed by dynamic linking

Move Away From Shared Library

- **Tweak: Do not link unnecessary shared libraries**
  - Check your linker options
- **Better Tweak: Use client-server model in place of shared libraries when possible**
  - From: m application processes linked to n shared libraries
  - To: Create a server process with n statically linked libraries. Use IPC mechanisms to connect m application processes to the server process

Toolchain Considerations

- **Plain old malloc allocator**
  - Two level memory allocation for user applications: malloc allocator on top of kernel page allocation
    - For performance reasons, malloc does not return all the free pages to kernel, it is done only when free memory exceed a threshold
      - These are dirty pages so kernel has no way to reuse it in a swapless system
  - Tweak: We can adjust parameters to make malloc allocator return unused pages to kernel more eagerly
    - libc: Call mallocopt function with M_TRIM_THRESHOLD as parameter
    - uClibc: No mallocopt function, change DEFAULT_TRIM_THRESHOLD at compile time
      - Default is 256K, too big for quite a few systems

Tools

- **Some proc entries**
  - Ignore free memory shown in /proc/meminfo
  - echo 3 > /proc/sys/vm/drop_caches followed /proc/meminfo is better
  - /proc/<pid>/maps for user processes
- **Tools from Matt Mackall**
  - /proc/kpagemap, /proc/<pid>/pagemap: Page walk tool
  - /proc/<pid>/smaps
    - Aware of shared memory pages
    - “Proportional Set Size (PSS)”: Divide shared pages by number of process sharing and attribute to each process
  - Matt Mackall's talks at ELC 2007, 2008, 2009
    - http://elinux.org/ELC_2009_Presentations
    - http://lwn.net/Articles/230975

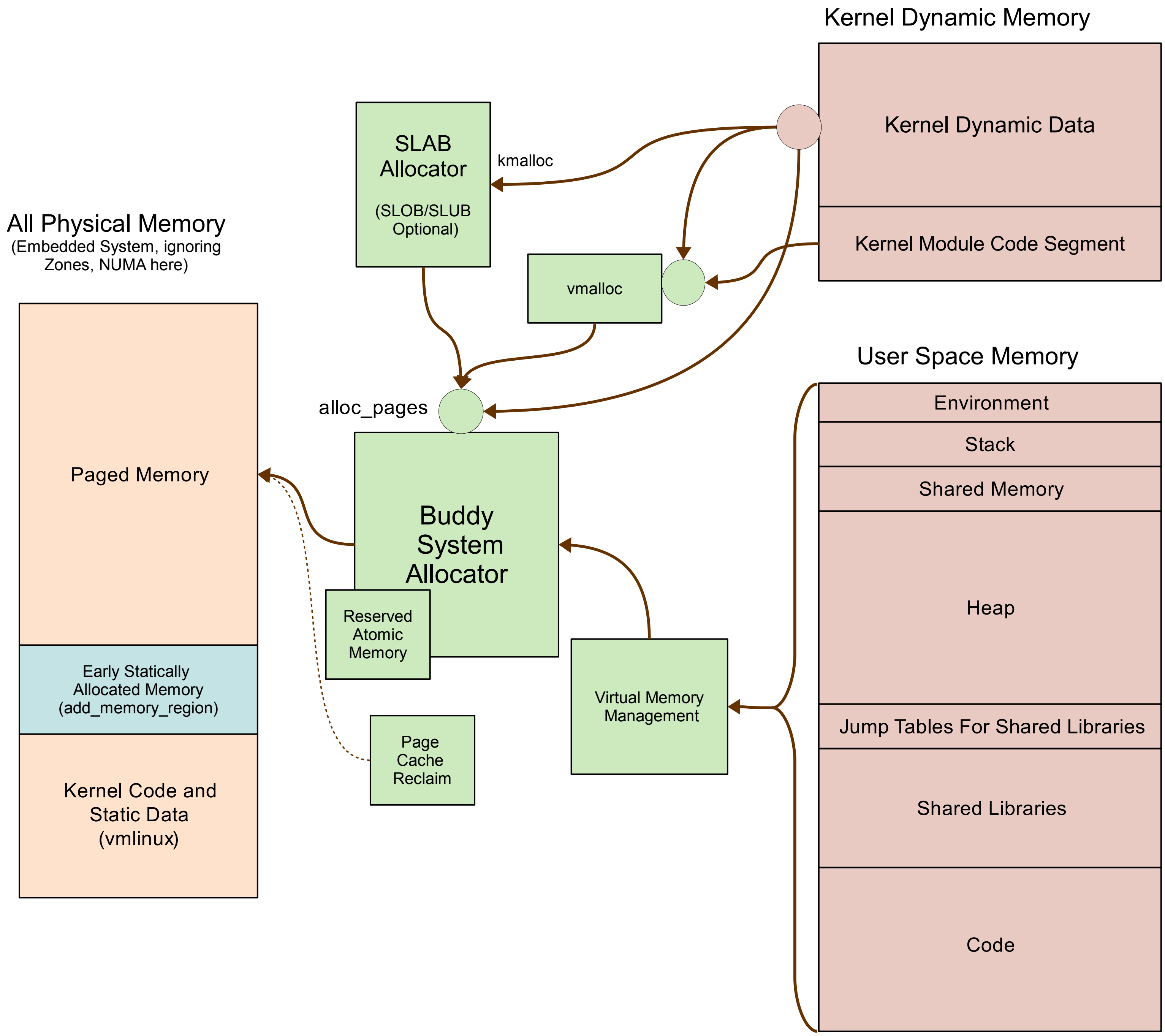
Tools

- **My (experimental) metric: Kernel Mobile Memory**
  - A number that simple math actually works
    - Kernel Mobile Memory = 5000k → kmalloc 100k → Kernel Mobile Memory = 4900k
  - Kernel Mobile Memory = # of memory pages that are free, or can be recycled by the kernel at anytime. For DRAM + Flash swapless system
    - Include: user application code segments (file backed pages that are not dirty), other cached files, etc.
    - Exclude: all dirty pages, memory consumed by kmalloc, user space malloc, etc.

Tools

- How to measure
  - Non-intrusive: Should be able to measure by walk through page table and get statistics
  - Intrusive: Let kernel memory reclaim mechanisms until it cannot find any free pages (super version of drop_caches), then do /proc/meminfo.
    - I wrote some code for older kernels, but it is not production-quality and it did not get updated for newer kernels
    - You can try to reuse power management code that puts the system into hibernation (pm_suspend_disk), as that part of kernel code does very similar things
- Use
  - If it is very low in absolute terms (for example, <1.5M), the system is about to lock up
  - If it is small compared to the working set (Don't have a way to measure:)), the system runs slowly

Revisit



Kernel Dynamic Memory
Kernel Dynamic Data
Kernel Module Code Segment
SLAB Allocator
(SLOB/SLUB Optional)
kmalloc
vmalloc
All Physical Memory
(Embedded System, ignoring Zones, NUMA here)
alloc_pages
Paged Memory
Early Statically Allocated Memory (add_memory_region)
Kernel Code and Static Data (vmlinux)
Buddy System Allocator
Reserved Atomic Memory
Page Cache Reclaim
Virtual Memory Management
User Space Memory
Environment
Stack
Shared Memory
Heap
Jump Tables For Shared Libraries
Shared Libraries
Code

Don't Forget the Basics

- **Squash + LZMA is usually an efficient read-only root memory system**
- **Remove unused symbols in shared libraries**
  - Debian mklibs script
    - http://packages.debian.org/unstable/devel/mklibs
- **Turn off unused kernel options**
- **Check all GCC options for your platform**
  - Choose an efficient ABI if available
- **These are low-level optimizations. Don't forget algorithm and data structure optimizations**
  - O(..) level improvements
  - Use arrays instead of linked list, trees when possible to avoid overheads

Conclusion

- **This may be tedious work, but if we avoid eyesight fixation or tunnel vision, it can be interesting too**
  - Memory consumed by kmalloc is important, but not everything. Application layer malloc, shared libraries need to be considered
  - Size is not everything. Fragmentation and low-memory CPU utilization are also important
  - It is interesting to find out that client-server model can be preferred for memory reasons