Boot Time Memory Management

Mike Rapoport
<rppt@linux.ibm.com>

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 825377

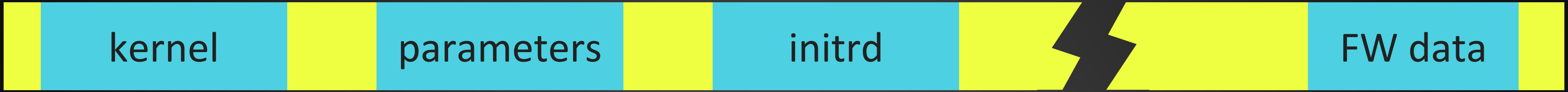
Topics

- Memory initialization
- memblock: API and internals
- From memblock to kmalloc

Even the physical page allocator … needs to allocate memory to initialise itself. But how can the physical page allocator allocate memory to initialise itself?

Mel Gorman, Understanding the Linux Virtual Memory Manager

Just before `_start`:



kernel parameters initrd FW data

- Lots of memory free
- Completely unclear where is it

Tread carefully!

From `_start:` to `mm_init()`

- Assembly sets up basic page table
  - Usually embedded into kernel .data
- `setup_arch()` continues memory initialization:
  - Detect physical memory
  - Reserve used areas (kernel image, initrd, firmware data etc)
- `start_kernel()` allocates several memory areas
  - Chunk size is usually larger than `MAX_ORDER`
  - Log buffer, VFS caches
- and calls `mm_init()`

And MM initialization is only a part of system init

Ancient history

- v2.0:

```
    setup_arch(&command_line, &memory_start, &memory_end);
    memory_start = paging_init(memory_start,memory_end);
    ...
    memory_start = console_init(memory_start,memory_end);
#ifdef CONFIG_PCI
    memory_start = pci_init(memory_start,memory_end);
#endif
    memory_start = kmalloc_init(memory_start,memory_end);
```

- v2.3.23pre3:
  - bootmem - a First Fit allocator which uses a bitmap to represent memory

From bootmem to memblock

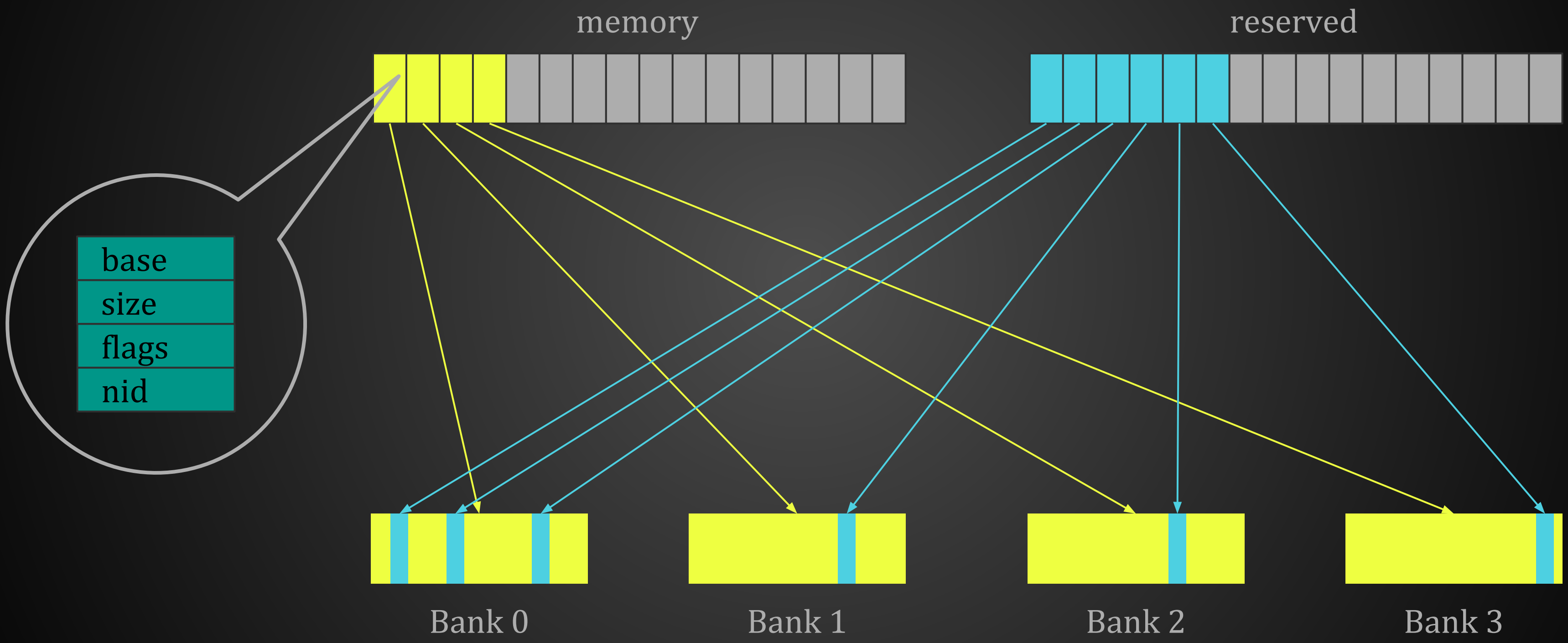
- Placing bootmem bitmap was challenging
  - Which bank, which NUMA node?
  - Bitmap size (1M for 32G of RAM)
- Slow transition to memblock
  - Started in v2.5 with powerpc64
  - Intermediate NO_BOOTMEM compatibility layer
- Completed in v4.20

Memblock vs bootmem

- + Static arrays instead of bitmap
  - ➢ Can be used before memory configuration is known
- + Arbitrary granularity instead of page

- – Allocation logic is more complex
  - ➢ But it’s ok, we should not have many of those anyway
- – Implicit growth of data structures
  - ➢ Too many early reservations may corrupt used memory

memblock structure



Basic APIs

- `memblock_add()`, `memblock_add_node()`
  - Register memory bank with memblock
- `memblock_remove()`
  - Make memory region invisible to the kernel
- `memblock_reserve()`
  - Mark used memory region as reserved
- `memblock_free()`
  - Mark memory region as free

Allocation APIs returning physical address

Functions that allocate memory and return its ***physical*** address:

- `memblock_phys_alloc()`
  - Allocate chunk of requested size with specified alignment
- `memblock_phys_alloc_range()`
  - Allocate a chunk within certain range
- `memblock_phys_alloc_try_nid()`
  - Allocate a chunk on a certain NUMA node

The memory is not cleared and may contain garbage!

Allocation APIs returning virtual address

Functions that allocate memory and return its ***virtual*** address:

- `memblock_alloc_try_nid_raw()`, `memblock_alloc_try_nid()`
  - Allocate chunk of requested size with specified alignment from certain NUMA node and within certain range
  - If constraints are too tight, try another node and then drop lower limit
- The “normal” variant clears the memory
- The ‘`_raw`’ variant does not!
  - Could be poisoned if VM debug is enabled

And their convenience wrappers

Functions that allocate memory, clear it and return its ***virtual*** address:

- `memblock_alloc()`
  - Allocate chunk of requested size with specified alignment
- `memblock_alloc_from()`
  - Allocate chunk above certain physical address
- `memblock_alloc_low()`
  - Allocate chunk in low memory
- `memblock_alloc_node()`
  - Allocate chunk in certain NUMA node

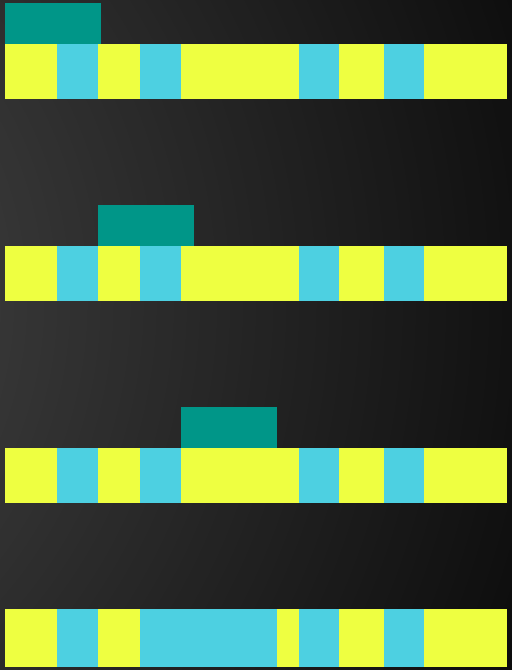
Convenience wrappers cont.

Function that allocates memory, and return its ***virtual*** address:

- `memblock_alloc_raw()`
    - Allocate chunk of requested size with specified alignment
- The memory is not cleared
    - Could be poisoned if VM debug is enabled

Under the hood

- `memblock_find_in_range_node()`
  - Find free area in given range and node
  - Traverses free memory areas
    - `memory && !reserved`
  - Can be top-down or bottom-up
- `memblock_reserve()` the area
  - Merge adjacent entries
  - Double `reserved` array if needed



Under the hood

- `memblock_alloc_range_nid()`
  - Try to find free memory with all the constraints
  - Retry on all nodes
  - Retry without mirroring requirement
    - Only on systems that support memory mirroring
  - Return ***physical*** address
- `memblock_alloc_internal()`
  - Try `memblock_alloc_range_nid()`
  - Retry without the lower bound
  - Return ***virtual*** address

Controlling memblock behaviour

- `memblock_allow_resize()`
    - Enable/disable resizing of memblock arrays
- `memblock_set_bottom_up()`
    - Set allocation direction (default is top-down)
- `memblock_enforce_memory_limit()`
- `memblock_cap_memory_range()`
- `memblock_mem_limit_remove_map()`
- `memblock_set_current_limit()`
- `memblock_trim_memory()`

Querying memblock state

- `memblock_phys_mem_size(), memblock_reserved_size()`
- `memblock_start_of_DRAM(), memblock_end_of_DRAM()`
- `memblock_is_memory(), memblock_is_reserved()`
  - Check for a given address
- `memblock_is_region_memory(), memblock_is_region_reserved()`
  - Check for a given range
- `memblock_get_current_limit()`
  - Get high limit for allowed allocations

Traversing memblock arrays

- `for_each_free_mem_range()`, `for_each_free_mem_range_reverse()`
  - Iterate over free memory areas
  - Take into account node and memory attributes
- `for_each_reserved_mem_region()`
  - Iterate over reserved memory areas
- `for_each_mem_range()`, `for_each_mem_range_rev()`
  - Iterate over intersection of memblock arrays
    - For example areas found in `memory` and absent in `reserved`
- `for_each_mem_pfn_range()`
- `for_each_memblock()`
- `for_each_memblock_type()`

Back to MM init

- Reserve used areas - `memblock_reserve()`
  - Kernel, initrd, firmware pages
- Detect and register physical memory - `memblock_add()`
  - Available banks, NUMA topology
- Set memblock parameters suitable for a machine
  - Limit to mapped memory, enforce bottom up allocations
- Use `memblock_alloc()` and friends to allocate memory
- Give pages to the buddy page allocator
  - `memblock_free_all()`
  - Usually called by arch specific `mem_init()`

References

- A quick history of early-boot memory allocators
- Boot Memory Allocator chapter
  - from "Understanding the Linux Virtual Memory Manager" by Mel Gorman
- Boot time memory management, kernel documentation

Thank you!