Memory management bits in `arch/*`

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

What is it about?

- Code and functionality duplication
- Hidden goodies
  - Some architectures are more equal than others
- arch <-> mm interface
  - Page table manipulation
  - Memory management initialization
  - Memory models
  - Memory hotplug

Short quiz

`pgd_offset(mm, address)`

returns a pointer to the entry in the top-level page table that maps the address.

Linux supports 25 architectures.

a) How many definitions of `pgd_offset` does Linux have?
b) How many does it really need?

Quiz answers

- `pgd_offset` is defined 31 times
- 28 of them are identical

```
#define pgd_offset(mm, addr)   ((mm)->pgd + pgd_index(addr))
```

  - arm64 and x86 have an additional helper:

```
#define pgd_offset_raw(pgd, addr)   ((pgd) + pgd_index(addr))
#define pgd_offset(mm, addr)   (pgd_offset_raw((mm)->pgd, (addr)))
```

  - s390 does its own magic:

```
static inline pgd_t *pgd_offset_raw(pgd_t *pgd, unsigned long address)
{ /* do stuff */ }
#define pgd_offset(mm, address) pgd_offset_raw(READ_ONCE((mm)->pgd), address)
```

Few more examples

- `pXd_index(addr)`

```
((addr) >> PXD_SHIFT) & (PTRS_PER_PXD - 1)
```

- `pXd_offset(pYd, addr)`

```
pYd_page_vaddr(*pYd) + pXd_index(addr)
```

- Early allocation
  - `alloc_page()` or `memblock_alloc()`
- VDSO mapping
  - Nearly identical code around `install_special_mapping()`

So where are the patches?

	5.0	5.3-rc1	-mm
`free_initrd_mem()`	26	8	6
`free_initmem()`	26	13	13
`pte_alloc_one_kernel()`	28	16	13
`pte_alloc_one()`	28	17	14

Why they are still so many?

- Actual differences
  - arm cleans D-Cache at pte_alloc
  - powerpc, sparc, s390 manage page tables in their own way
- Testing is not trivial even when possible
  - non-trivial cleanups are scary
- (Ab)use for extensions beyond “default” functionality
  - Change protection bits of various mappings
  - Change chassis LED state
  - Verify errata fix availability

What's next?

- Page table management
  - New page table manipulation API?
    - https://lore.kernel.org/lkml/20180424154355.mfjgkf47kdp2by4e@black.fi.intel.com/

  or
  - Move `p?d_offset(),p?d_index()` etc to `asm-generic`
  - Deal with multiple variants of early allocation
- VDSO etc
  - Generalize what's possible and spread the goodness around
- Memory initialization
  - Major surgery?

DISCONTIGMEM

- alpha
  - Switching to SPARSE seems simple
- arc
  - Creative use to enable high memory below low memory
  - Supposedly more efficient than sparse
- ia64
  - DISCONTIGMEM is required for SPARSE and custom VMEMMAP
- m68k
  - Hard to define appropriate section size for SPARSE
  - VMEMMAP is resource greedy
- mips
  - `default y if SGI_IP27`, anybody has SGI Origin to test?

Arch hooks in mm initialization sequence

- `setup_arch()`
  - From ~200 lines to lots of black magic
- `mem_init()`:
  - Give memory to page allocator and print memory info
- `free_initrd_mem()`
  - Free initrd memory
- `free_initmem()`
  - Free memory in .init.* sections

`free_initrd_mem()`

Only few remain:

- arm and ia64 have custom definitions of initrd boundaries
- arm64 calls `memblock_free()`
  - But why?
  - Should others call it as well?
- x86 changes memory protection
  - Could be useful for other architectures

free_initmem()

- Changes of .init.* areas protection
  - Move to `free_initmem_default()`
- Last-time callback into arch code
  - Add `arch_finalize_boot()` callback

```
diff --git a/init/main.c b/init/main.c
@@ -1114,10 +1114,10 @@ static int __ref kernel_init(void *unused)
     free_initmem();
-    mark_readonly();

     /*
      * Kernel mappings are now finalized - update the
      * userspace page-table to finalize PTI.
      */
     pti_finalize();
+    arch_finalize_boot();

     system_state = SYSTEM_RUNNING;
```

`setup_arch()`

- Reserve used areas
  - Kernel, initrd, firmware pages
- Detect physical memory
  - Available banks, NUMA topology
- Build kernel page tables
  - Linear map
- Initialize memory map
  - `struct page` array(s)
- Calculate zone limits

mem_init()

- Set `max_mapnr` and `high_memory`
- Give pages to the buddy page allocator
  - Different code paths for low and high memory
- Print memory info

```
Memory: 7824920K/8077548K available (8618K kernel code, 1335K rwdata, 4032K rodata,
1488K init, 1284K bss, 252628K reserved, 0K cma-reserved)
```

- Architecture-specific initializations
  - Hypervisor hooks, build-time consistency checks, zero page setup
  - Some must run before page allocator is up, other afterwards

A bit of context

```
asmlinkage __visible void __init start_kernel(void)
{
        ...
        pr_notice("%s", linux_banner);
        setup_arch(&command_line);
        ...
        build_all_zonelists(NULL);
        page_alloc_init();
        ...
        mm_init();                  /* calls mem_init() */
        ...
}
```

Assumptions

- Memory detection is about conversion of firmware data to memblock
- Early NUMA topology detection is possible
- Nothing uses `struct page` before page allocator is up

RFC: update mm init sequence

- Make memory reservation and detection explicit
  - Move them before `setup_arch()`
  - Use architecture-specific memblock sizing if needed
- Move memory map initialization to generic code
  - Make `free_area_init_nodes()` available to non-NUMA machines
  - Add `get_zone_limits()` callback
  - Fold `sparse_init()` and friends into `free_area_init_nodes()`
- Add arch callback just before mm_init()
  - And maybe another one just afterwards

RFC: reserve and detect memory early

```
diff --git a/init/main.c b/init/main.c
@@ -595,4 +595,6 @@ asmlinkage void start_kernel(void)
        pr_notice("%s", linux_banner);
+       early_reserve_memory();
+       detect_memory();
        setup_arch(&command_line);
        mm_init_cpumask(&init_mm);
        setup_command_line(command_line);
```

RFC: initialize memory map in generic code

```
diff --git a/init/main.c b/init/main.c
@@ -546,13 +546,15 @@ static void __init
report_meminit(void)
 /*
  * Set up kernel memory allocators
  */
 static void __init mm_init(void)
 {
+    arch_pre_mm_init();
+    memmap_init();
     /*
          * page_ext requires contiguous pages,
          * bigger than MAX_ORDER unless SPARSEMEM.
          */
     page_ext_init_flatmem();
     report_meminit();
     mem_init();
     kmem_cache_init();
```

RFC: dissolve mem_init()

```
diff --git a/init/main.c b/init/main.c
@@ -526,4 +526,6 @@ static void __init report_meminit(void)
       const char *stack;

+      mem_init_print_info(NULL);
+
       if (IS_ENABLED(CONFIG_INIT_STACK_ALL))
            stack = "all";
@@ -556,6 +558,7 @@ static void __init mm_init(void)
        */
       page_ext_init_flatmem();
+      memblock_free_all();
+      free_highmem_pages();
       report_meminit();
-      mem_init();
       kmem_cache_init();
       pgtable_init();
@@ -566,4 +569,5 @@ static void __init mm_init(void)
       init_espfix_bsp();
       pti_init();
+      arch_post_mm_init();
 }
```

Challenges

- Hard to get feedback from less active architectures
- Non-trivial changes are scary
- No consideration for neighbours for new arch/ code

Thank you!

mm initialization

- `setup_arch()`
  - lots of black magic
- `setup_per_cpu_areas()`
  - Overridden by several architectures
  - Relies on functional `for_each_possible_cpu()`
- `build_all_zonelists()`
  - Expects NUMA topology
- `page_alloc_init()`
  - initialize page allocator^w^w^w setup page allocator callbacks for CPU hotplug
- `mm_init()`
  - Actually init page allocator, kmem caches and vmalloc