Design of the Linux Percpu memory Allocator

Prathu Baronia

16-03-2024

$whoami

- Linux user for past 8 years and started diving in kernel 4 years back.
- Contributed some patches in `mm` subsystem upstream.
- Used to work in Oneplus Kernel and Qualcomm CPU team.

Before we begin

- Kernel version considered: v6.8

What are percpu variables & why I need it?

- Special variables where we need to allocate per cpu instances
- No locking needed since its specific to a cpu
- Example

```
// Records register data for each cpu
DEFINE_PER_CPU(struct cpuinfo_arm64, cpu_data);
```

- Also used in the kernel for percpu counters, percpu page caches etc.

What this talk does not cover

- Handling of percpu allocations in modules
- Milestones in the development history of the percpu allocator

Dev APIs

Static API

- `DEFINE_PER_CPU(type, name)`
- Access the percpu variables with `per_cpu()` and `this_cpu_ops` macros.
- `per_cpu(var, cpu)` will return you the instance of var for given cpu.
- `per_cpu_ptr(var, cpu)` will return you ptr to the instance of var for given cpu.

Dynamic API

```
void __percpu *alloc_percpu(type)
free_percpu(void __percpu *ptr)
```

- Access the variables the same way. :)

Under the hood of static APIs

```
#define DEFINE_PER_CPU(type, name)
    DEFINE_PER_CPU_SECTION(type, name, "")

#define DEFINE_PER_CPU_SECTION(type,name)
    __PCPU_ATTRS(sec) __typeof__(type) name

#define __PCPU_ATTRS(sec)
    __percpu __attribute__((section(PER_CPU_BASE_SECTION sec)))

#define PER_CPU_BASE_SECTION ".data..percpu"
```

Linker magic

- From include/asm-generic/vmlinux.lds.h

```
#define PERCPU_SECTION(cacheline)
    . = ALIGN(PAGE_SIZE);
    .data..percpu    : AT(ADDR(.data..percpu)) {
        PERCPU_INPUT(cacheline)
    }
```

- This is placed in the range [__init_begin, __init_end] which is freed after init by free_initmem().

Linker magic

- From include/asm-generic/vmlinux.lds.h

```
#define PERCPU_INPUT(cacheline)
    __per_cpu_start = .;
    *(.data..percpu..first)
    . = ALIGN(PAGE_SIZE);
    *(.data..percpu..page_aligned)
    . = ALIGN(cacheline);
    *(.data..percpu..read_mostly)
    . = ALIGN(cacheline);
    *(.data..percpu)
    *(.data..percpu..shared_aligned)
    PERCPU_DECRYPTED_SECTION
    __per_cpu_end = .;
```

Under the hood of dynamic APIs

```
#define alloc_percpu(type)
    (typeof(type) __percpu *)__alloc_percpu(sizeof(type),
                                            __alignof__(type))

void __percpu *__alloc_percpu(size_t size, size_t align)
{
    return pcpu_alloc(size, align, false, GFP_KERNEL);
}
EXPORT_SYMBOL_GPL(__alloc_percpu);
```

Lets talk about design of the allocator . . . But wait

- We first need to talk about its dependencies
- And things that depend on the percpu allocator

Memblock Allocator

- Allocator used before normal allocators are up.
- Provides APIs like `memblock_alloc*` which can allocate and manage memory early in the boot process with NUMA support.
- `arm64_memblock_init()` & `mem_init` are two important functions to look at.
- `CONFIG_ARCH_KEEP_MEMBLOCK` controls whether memblock data structures are freed or not after system initialization.

Generic NUMA support

- Used by `arm64` & `riscv`
- `arch_numa_init()` has both device tree and acpi support.
- Parses device tree for cpu and memory nodes to collect information about system organization.
- Provides functions to calculate distance(i.e. memory latency) between NUMA nodes and a node to cpu map which will come handy in the percpu allocator.
- Later in `start_kernel()->setup_per_cpu_areas()` (Entry point in the allocator)

Now onto the allocator

- `mm/percpu.c`
- Allocation using chunks and units.

```
 c0                           c1                         c2
 -------------------          -------------------        ------------
| u0 | u1 | u2 | u3 |        | u0 | u1 | u2 | u3 |      | u0 | u1 | u
 -------------------  ......  -------------------  ....  ------------
```

`setup_per_cpu_areas()`

- Generic NUMA version of the above function calls into the allocator to setup the first chunk.
- There are two ways to setup the first chunk:-
  - `pcpu_embed_first_chunk(...)`
  - `pcpu_page_first_chunk(...)`
  - These are controlled by `percpu_alloc` cmdline param.
- Also suprisingly UP systems will also have a `setup_per_cpu_areas()`.

Page mapping the first chunk

```
commit 09cea6195073ee1d0f076d907d9249045757245d
Author: Kefeng Wang <wangkefeng.wang@huawei.com>
Date:   Fri Nov 5 13:39:44 2021 -0700

Percpu embedded first chunk allocator is the firstly option, but it
could fails on ARM64, eg,

percpu: max_distance=0x5fcfdc640000 too large for
vmalloc space 0x781fefff0000

then we could get
WARNING: CPU: 15 PID: 461 at vmalloc.c:3087 pcpu_get_vm_areas+0x488/0x838
and the system could not boot successfully.
```

Static allocations

- Static allocations are handled by the first chunk which is organized as:-
    - *<Static | Reserved | Dynamic>*
- Reserved section corresponds to static percpu variables from modules.
- And Dynamic section takes care of normal runtime allocations.

After embed|page setup

- `pcpu_setup_first_chunk()` is called by both the variants after copying the static area.
- First chunk is served by two more chunks corresponding to the reserved and dynamic areas.
  - `pcpu_reserved_chunk` & `pcpu_first_chunk`(badly named IMO)
- Initialize `__per_cpu_offsets[]` which is used to calculate per cpu addresses of variables.

Chunk management

- All chunks are organized into lists in ascending order of free sizes.
- All chunks are managed by a bitmap with metadata blocks.
- Each metadata block has scanning and contiguous area hints which help to avoid iteration over large portions of bitmap.
- Chunk management functions like creation and population has two versions `mm/percpu-vm.c` * `mm/percpu-km.c`
  - `percpu-vm.c` is the default allocator.
  - `percpu-km.c` is for `nommu` archs.

Dynamic allocation and freeing paths

Allocation

- Allocator tries to allocate from the fullest chunks first.
- Finds the offset within a chunk which can fit the size and alignment requirement and allocates the area and returns a percpu ptr.
- If there is no chunk available which can fulfill the requirement we try to create a new chunk.

Freeing

- Locates the chunk which corresponds to the given ptr.
- `percpu_free_area` finds the size of the allocation using the boundary bitmap and clears the allocation map.
- Both paths and chunk movement on lists controlled under the spinlock `pcpu_lock`

Few things that I have not touched upon here

- Reclaiming of chunks
- Hint management inside the chunks

this_cpu_ops

```
int *y;                        |
int cpu;                       |
cpu = get_cpu();               |     this_cpu_inc(&x);
y = per_cpu_ptr(&x, cpu);      |
(*y)++;                        |
put_cpu();                     |
```

Thanks for attending!

- Any questions?
- P.S: I am looking for a job. Any openings? :D