

Lecture 7: March 23

*Lecturer: Alessandro Pellegrini**Scribe: Anzhelo Xhebraj*

Spawning many processes with a short lifetime it is not performed efficiently through the buddy system.

Quicklist avoid contention of allocating pages by pre assigning pages to cores. The SLAB allocator is used for buffers.

In quicklists no synchronization is needed. If no entries available in the list it asks memory to the buddy system through `_get_free_page`

7.1 Quicklist Allocation

Kernel threads can move around in cores: `get_cpu_var` is a wrapper to a set of calls that disables preemption for that specific core until re enabled by `put_cpu_var`.

7.2 SLAB Allocator

There is a list of caches where each cache keeps a slab of a specific size. A cache is organised in three types of slabs: full (Don't have buffers to be given), partial (some used and some free), free (deallocated or never allocated). Objects are abstractions over pages.

7.2.1 SLAB Interfaces

Found in `linux / malloc.h`. `kmalloc` asks for a given size and returns the virtual address of a buffer of that size. `kfree` frees memory allocated via `kmalloc`. `k_malloc_node` is an api to the slab allocator that in the end will ask to the buddy system of a specific numa node for allocating some page.

`kmalloc` should be used for frequent allocations and deallocations of the same size.

up to kernel v 3.9.11 there was the `struct cache_size`. `cs_cachep` is a pointer to the memory. There is a table of multiple size fixed-size caches.

In kernel 3.10 we move from fixed size to list with spinlock again. You can either have a shared across cores allocator or one allocator for each core. Having one allocator for each core requires space. What's the difference btw using the buddy system and slab? buddy has one spinlock for each numa node while slab has one spinlock for each size of cache.

SLAB and Buddy both are for the kernel

Per node cache coloring: size of object then padding and so on. Why is coloring used? to align objects to L1 Cache Bytes. Two objects of the same size will not fall in the same cache line. An object of size greater than one line is padded to the size of multiple cache lines.

This ensures that two slabs objects allocated will not fall in the same cache line to not fall into cache contention.

Members that are accessed together and used frequently together (Common Members) are placed close together to optimise cache hits for example the spinlock and slab partial in `kmem_cache_node_struct`.

(Loosely related fields): due to the false cache sharing problem we have that cache controllers in order to be coherent tell the others cache controllers that they are going to write that line wanting "mutual exclusion". With coloring we ensure that different buffers will not fall in the same cache line.

7.2.2 Cache flush operations

Similarly to the TLB relies on the hardware specific operations for granularity and coherency of the flushing. There are also problems because the hardware cache uses virtual addresses, therefore two processes addressing the same virtual address might have a cache hit to a region of memory that is not the physical one they wanted to access. After flushing the page cache we must also flush the TLB.

flush_cache_all: Flushes the entire CPU cache system. It is used for when global data structures, for example kernel page tables, are changed to ensure cache coherency.

others...

What is the best way to devise a cache? physical or virtual? Intel architectures use Virtual addresses to tag L1 cache. If there is a miss in L1 the TLB is consulted to get the physical and check the L2 which is addressed through the physical address. There is a protocol btw L1 and TLB to know whether a virtual address is consistent with the current paging scheme. Therefore in intel we do not care about cache consistency.

Virtual aliasing is the problem described above where we tag the L1 cache through virtual addresses but if the cache is not coherent ...

The other apis are a low level api that are used by the description above.

copy from and to user ensures that the copy of memory is done correctly since there might be a process switch and the write might be writing memory of another process.

Access ok checks whether the memory area passed is correctly mapped to that process.

vmalloc used to map some memory in the kernel in a stable way, that will be used for a long time. No idea about the memory contiguousness. No info about the organisation of physical frames. Used for usually loading some kernel module for code, data etc of the module. It doesn't rely in either the Buddy system or slab.

virt_to_phys and viceversa used in **kmalloc** or get free page to compute the mapping btw physical and virtual addresses. This is done to be hardware independent when developing a kernel module to not rely on offsets etc.

For allocation size in **kmalloc** is limited to 8KB in Linux. **vmalloc** btw 64/128MB. **Kmalloc** is physical contiguous while **vmalloc** no. **Vmalloc** invalidates transparently the TLB etc while **kmalloc** no. This is done because for example in loading kernel modules you want that all threads have visibility of the change.

After setting up all the memory **trap_init()** initialises the IDT and GDT. The only entry point to kernel land from user space is through the interrupt 0x80. The same is done in Windows.

That interrupt executes the system call dispatcher which finds out what the user code wants to do. Every system call has a code assigned to it.

Interrupts automatically reset the Flags while Traps do not. The dispatcher explicitly clears interrupts through **cli** instruction. In multi-core systems this is not enough, we must

ensure correctness of all the data structures of the kernel. Spinlocks (implemented through the `cmpxchg` instruction) are used to access data structures.

Since syscalls have preassigned numbers if we change only one of them we break backward compatibility.

Macros are used for generating asm volatile blocks defining a syscall. Multiple macros depending by the number of the parameters of the syscall (usually at most 6 parameters).