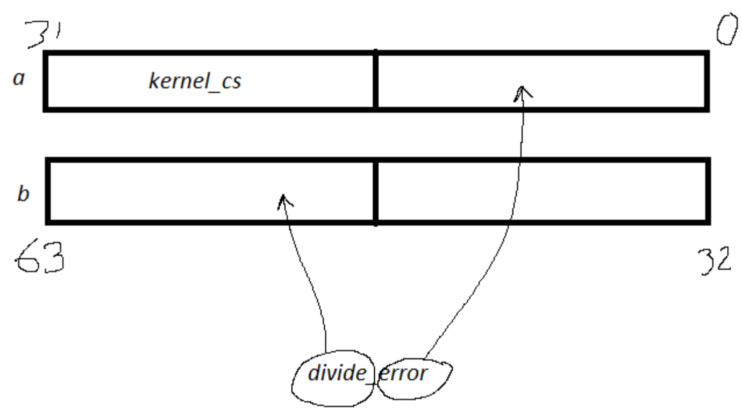
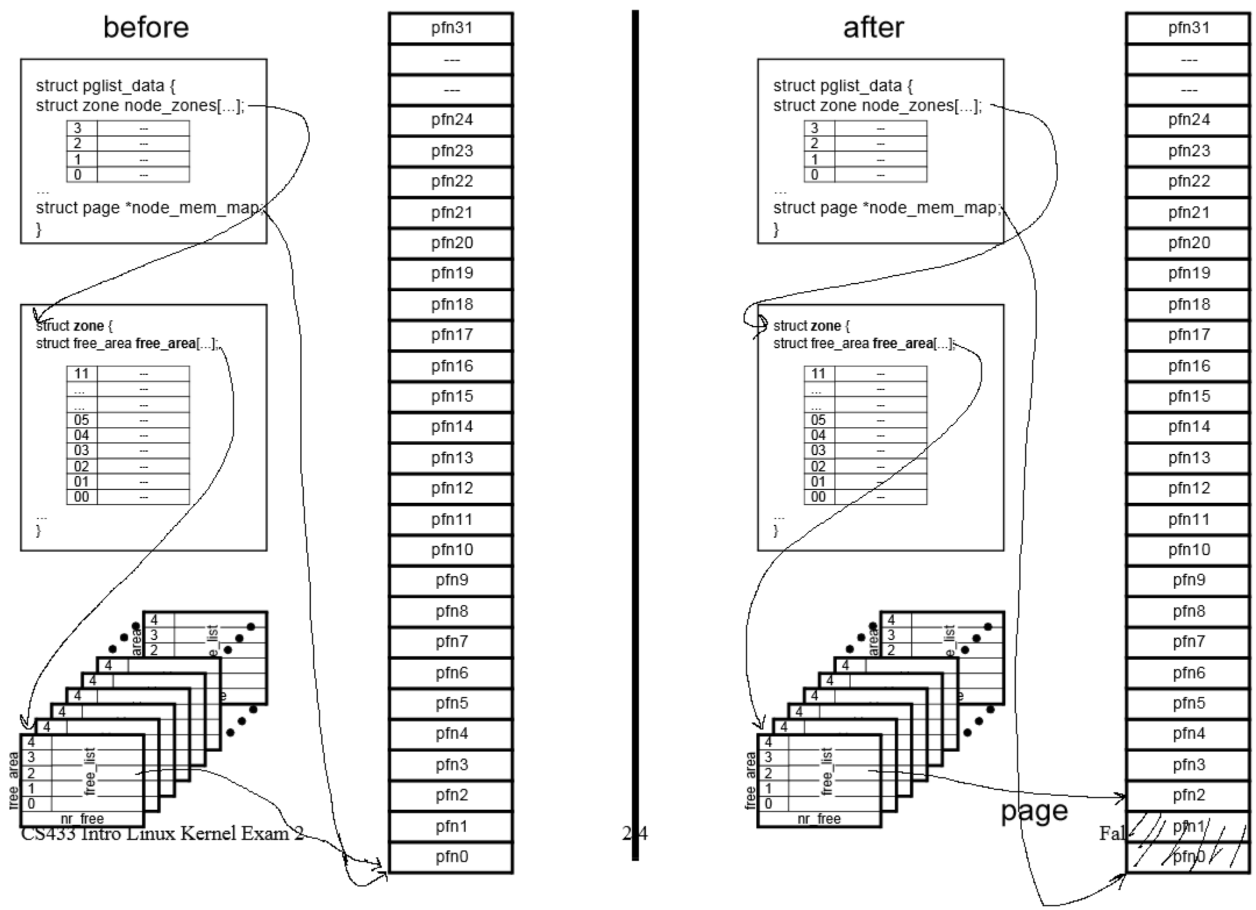
```c
#else
static inline void pack_gate(gate_desc *gate, unsigned char type,
                             unsigned long base, unsigned dpl, unsigned flags,
                             unsigned short seg)
{
        gate->a = (seg << 16) | (base & 0xffff);
        gate->b = (base & 0xffff0000) | (((0x80 | type | (dpl << 5)) & 0xff) << 8);
}

#endif
```



Problem 2 (Memory buddy system - 30 points): Assuming all 32 pfns are free, show what happens before and after calling **page = alloc_pages(gfp_mask, 1);** from the *movable* region of *zone DMA*. Ignore gfp_mask. Make all the necessary connections to indicate free/allocated pages before and after. Indicate where **page** and **node_mem_map** point. Connections must be made to specific array entries to be considered accurate.



Problem 4 (Processes - 25 points)

(a) Fill in the table with the first five context switches based on the version we discussed in class: use the following convention: p0=swap-per,p1=kernel_init, p2=kthreadd, p3=migration,p4=ksoftirqd

Switch no	PID to switch from	Functions in which the switch occurs - List a sequence of at least three functions that can identify where the switch takes place.	Processes in the RB tree	PID to switch to
1	P0	*1. copy_process* *2. wake_up_new_task* *3. put_pid*	P1, P2	P1
2	P1		P1, P2	P2
3	P2		P1, P2, P3	P1
4	P1		P1, P2, P3	P2
5	P2		P1, P2, P3, P4	P1

(b) Indicate the program statement(s) at which context switches ultimately occur. Be as precise as you can.

```
45        "movl %[next_sp],%%esp\n\t"        /* restore ESP    */ \
```

Problem 4 (20 points, the first three kernel threads):
What is the name, not number, of the very first kernel thread?

sched/swapper

What is the name of the function that calls two functions of the same name but with different parameters to create the first two kernel threads?

kernel_thread

What are the process numbers and names of the two kernel threads created by the very first kernel thread?

P1 - init thread
P2 - kthreadd

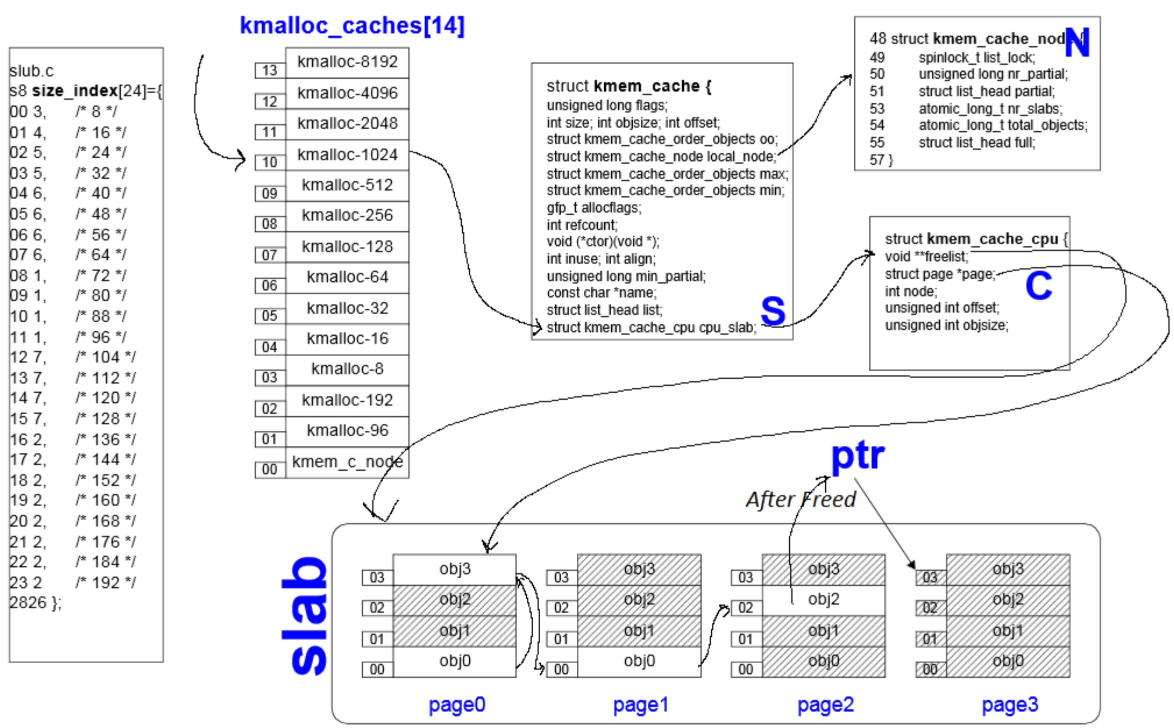
After the two kernel threads described above are created, what happens to the very first kernel thread and *where*?

sched handles irq scheduling and stays in kernel space which is invisible to users

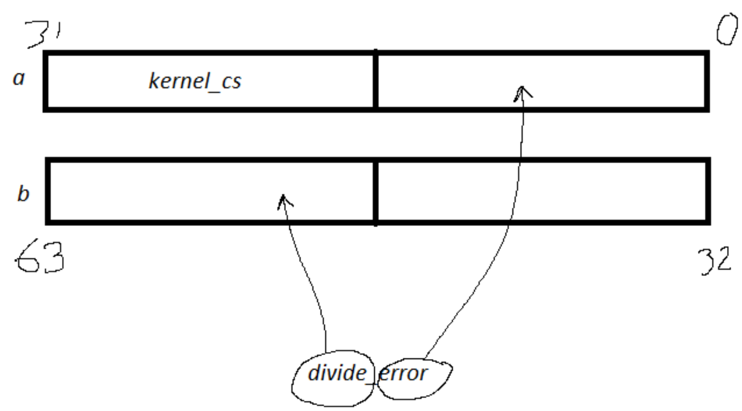
Problem 3 (Cache - 30 points): Given the cache below, show what happens *after* the two following C statements are executed by the kernel.

```
int ptr[200]; kfree(ptr);
```

Ignore flags. Make necessary connections such that ptr is returned to the slab for later use. Assume that the free objects are linked and available in ascending order of page numbers as well as object numbers.



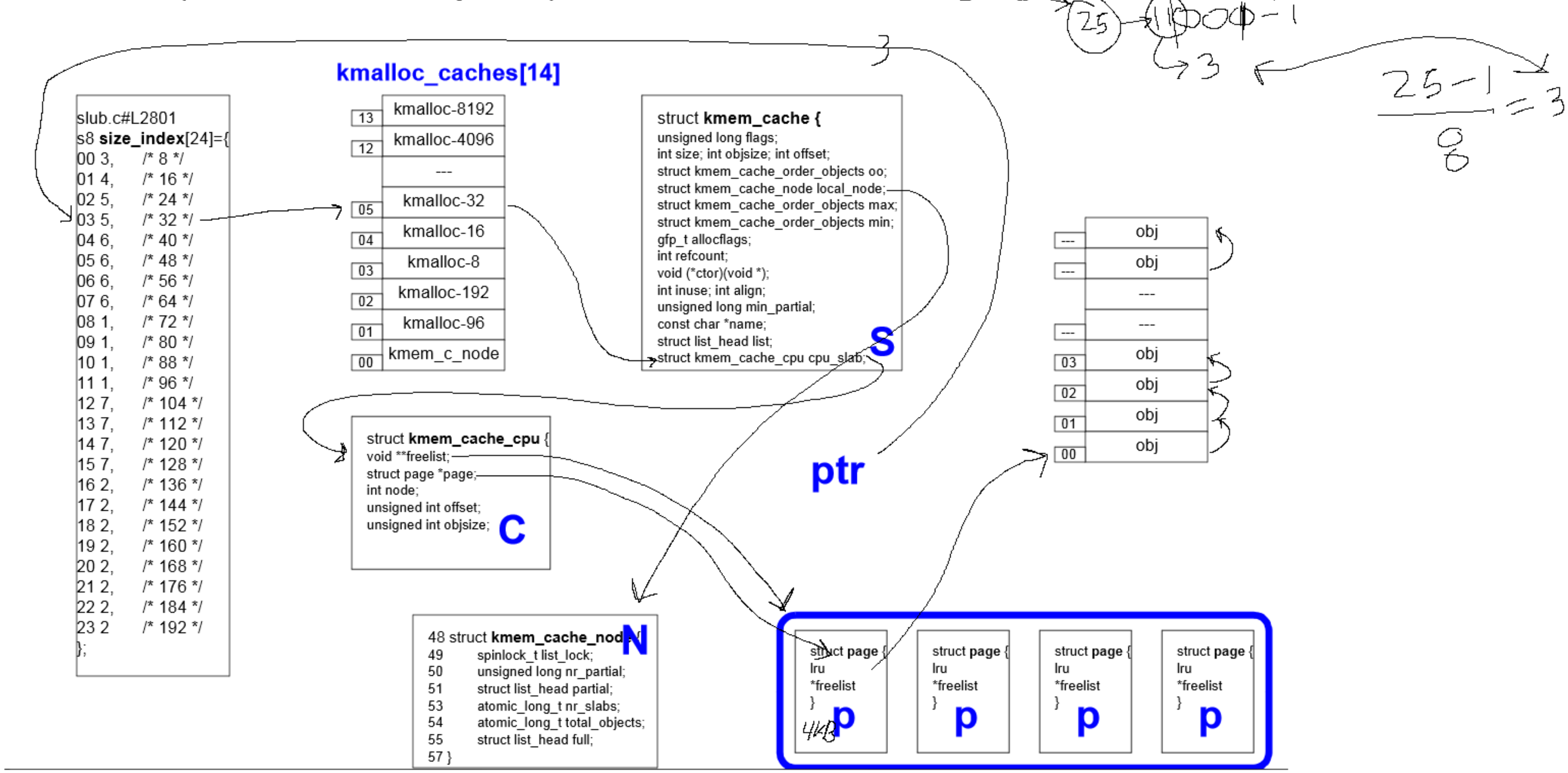
```
#else
static inline void pack_gate(gate_desc *gate, unsigned char type,
                             unsigned long base, unsigned dpl, unsigned flags,
                             unsigned short seg)
{
        gate->a = (seg << 16) | (base & 0xffff);
        gate->b = (base & 0xffff0000) | (((0x80 | type | (dpl << 5)) & 0xff) << 8);
}

#endif
```

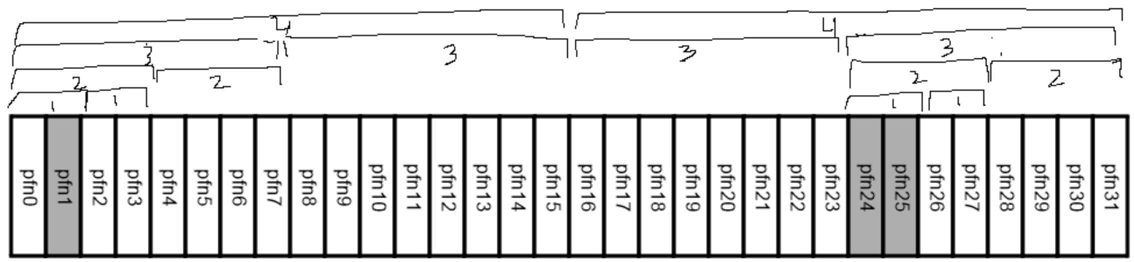


Problem 3 (Cache - 25 points): Assuming the cache is completely empty, show what happens after calling

```
   ptr = kmalloc(sizeof("CS433 Intro Linux Kernel\n"),flags);
```

Ignore flags. Make as many necessary connections as you can such that **ptr** ~~points to the desired memory piece~~ while the remaining objects can be accessed instantly. Connections must be made to specific array entries to be considered accurate. Start from ~~size_index[]~~





order	page_idx	combined_index	buddy
0	1	0	0
1	24	24	26
2	24	24	28
3	24	24	16
4	16	16	0
5	—	—	—

order	number of free page groups	
	before	after
0	1	1
1	2	1
2	2	1
3	2	1
4	0	1
5	0	0

Problem 2 (Memory - Buddy system - 25 points): Consider a snapshot of the memory system consisting of 32 pfns below. Show what happens after freeing a group of 2 pages, pfns 24 and 25, by filling in the two tables below.