1. Deploy a module inside kernel in following ways:
   1. Dynamic
      1. Modeprobe
      2. Insmod
   2. Static linking
2. Why is kernel mapped to process address space?
   1. Kernel is said to be in the process address space even when a process cannot address the kernel directly.
   2. When the process makes a system call, we don't need to switch the page tables (from process address space page table to kernel address space page table) for servicing the system call (which should be done only in kernel mode).
   3. Each process has entries of kernel’s page tables in its page table. Some kernel events which won't run in process context will load the page tables only for kernel.
   4. User process’s inability to peek and poke the kernel code and data is not due to different address spaces, it's due to different access rights/permissions set in the page tables. Kernel pages are set up in such a way that regular processes can't access them.
   5. It is quicker to change the security level (thus allowing access to the pages that are otherwise protected, as mentioned in Alexey's answer) in order to perform system calls and other kernel provided functions than it is to change the security level and the entire virtual memory map, along with all the associated TLB cache flushes and everything else involved in a full context switch.
3. Difference between mutex and spinlocks?
   1. On single core: mutex is better as one thread can sleep and other can do some task. The sleeping thread will not be scheduled at all the.
   2. On multiple cores: spinlock is better as a thread could be scheduled on some other core and can run for its quantum spinning for the lock to be released. Putting a thread to sleep and waking it up is usually expensive.
   3. Hybrid locks are mutex locks in which a thread will spin for a short duration so that if the lock is released within that short duration, it will not have to go to sleep and then waken up.
4. Page fault:
   1. Valid virtual addresses are mapped to Physical addresses in MMU. Done via pages.
   2. Program accesses virtual memory. If page is present, then good, else:
      1. Valid page fault: MMU will raise exception to kernel that a physical page is required for the virtual page accessed. Then page will be read into memory by kernel.
      2. Invalid: kernel in this case will generate segfault for the process.
   3. How the program is loaded page by page? Suppose a program has 10 functions and all are not called at the same time. Then pages could be allocated for the functions that are being called at a given time, and the functions called later could be provided memory later for their stack.
5. Interrupts and Exceptions.
   1. Devices require a prompt response from the CPU when various events occur, even when the CPU is busy running a program.