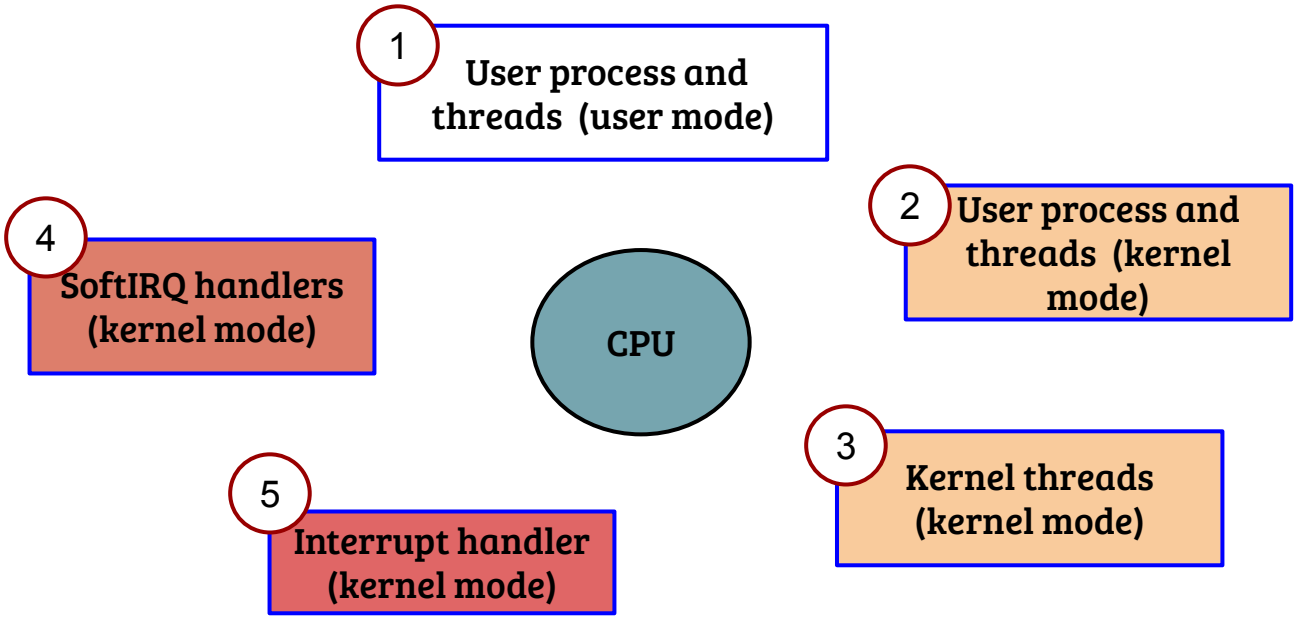
CS614: Linux Kernel Programming

Kernel Threads

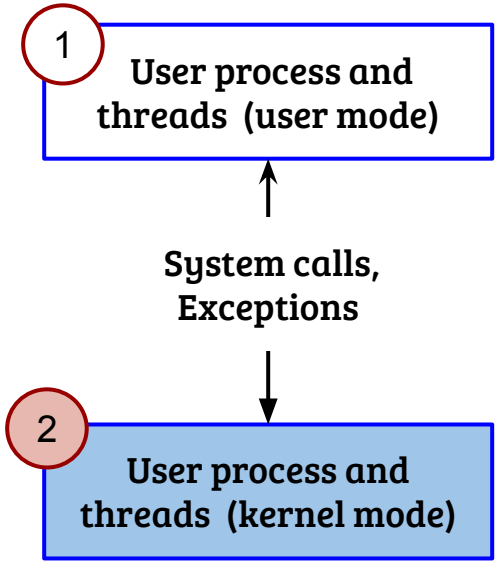
Debadatta Mishra, CSE, IIT Kanpur

Recap: Execution contexts in Linux



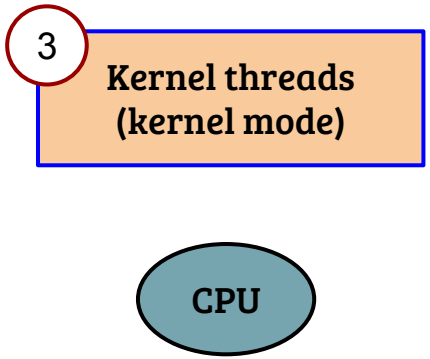
- In a linux system, the CPU can be executing in one of the above contexts
- For (3), (4) and (5), the context is not associated with any user process

Recap: User contexts



- What are the differences in execution states?
- What is the exact entry and exit mechanisms—user to kernel context switch and vice-a-versa?
- What is the need for save and restore of the execution states? How implemented in Linux kernel?
- Access/modification of user execution state from the kernel mode, how?
- How OS manages the user contexts? (scheduling to be covered)

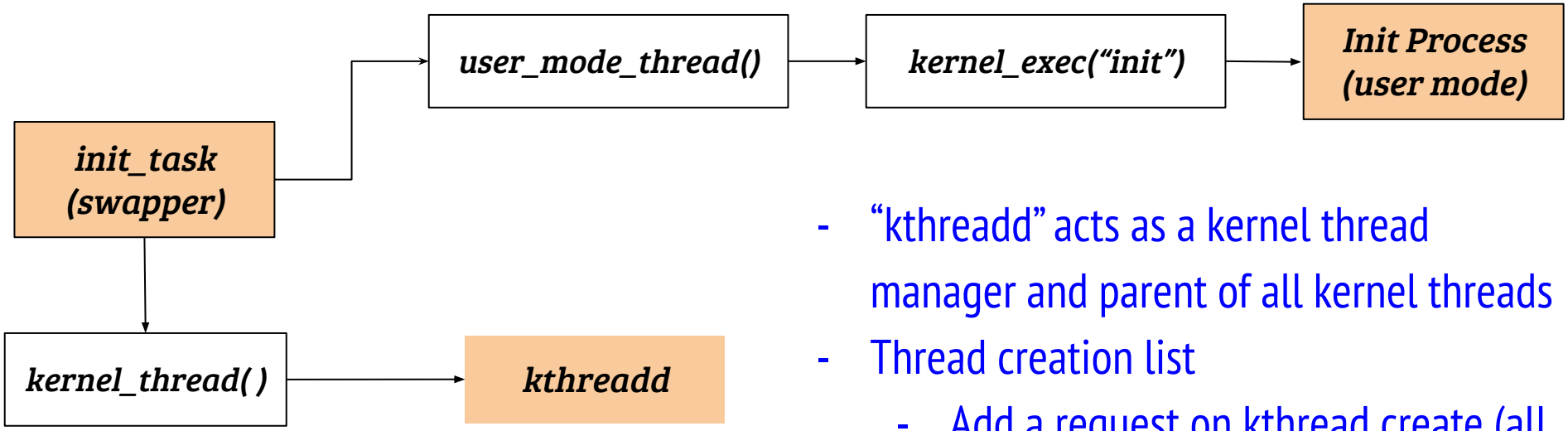
Kernel threads



- Kernel threads are independent of user processes and threads
- How kernel threads are created and managed?
- How is a kernel thread task is different?

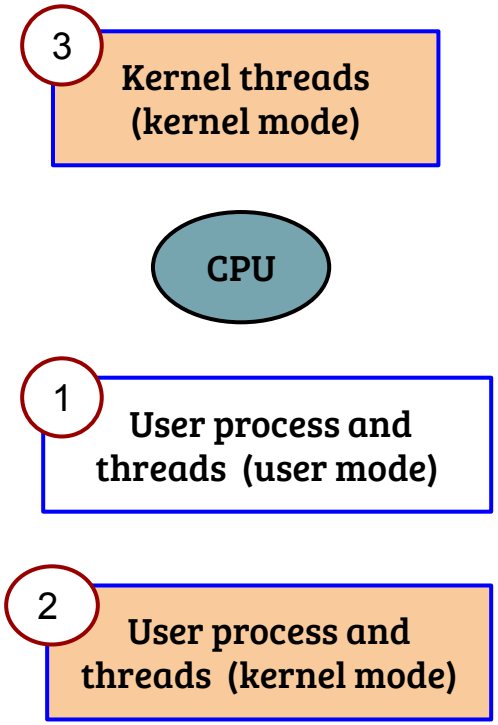
Revisiting the first process creation

- How kernel threads are created and managed?



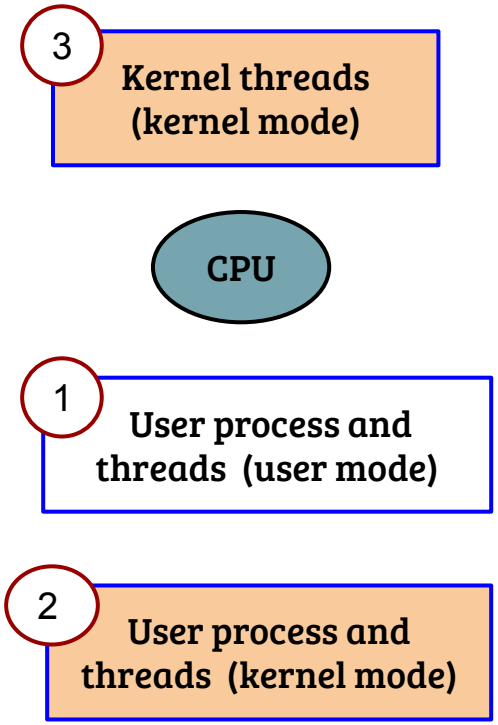
- "kthreadd" acts as a kernel thread manager and parent of all kernel threads
- Thread creation list
  - Add a request on kthread create (all types of kernel threads)
  - Wakeup kthreadd
  - Kthreadd ⟶ kernel_thread( )

Kernel threads



- How is a kernel thread different?
  - Kernel thread never executes in user mode
  - Does not require a MM context of its own
  - Generally treated with same priority as that of user processes for scheduling
- Which page table is used by any kernel thread?

Kernel threads



- How is a kernel thread different?
  - Kernel thread never executes in user mode
  - Does not require a MM context of its own
  - Generally treated with same priority as that of user processes for scheduling
- Which page table is used by any kernel thread?
  - It can use the page table of outgoing task
  - What is the catch?