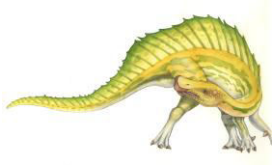
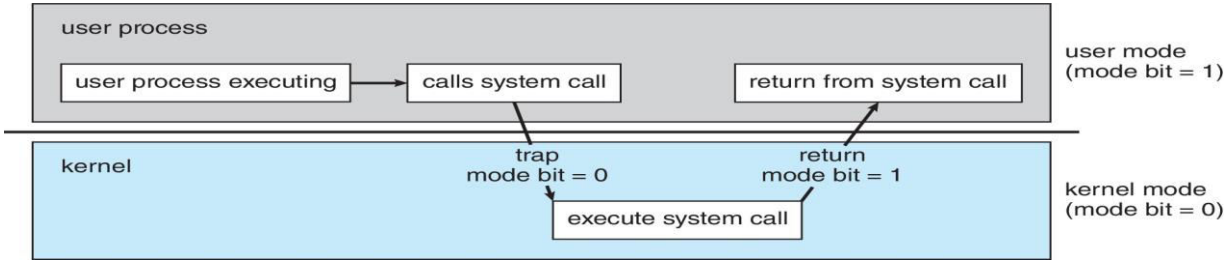
Review questions on processes and threads

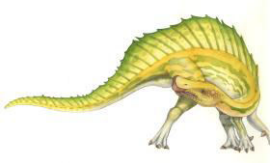


User and kernel space

- To obtain services from the operating system, a user program must make a **system call**, which invokes the kernel.
- In order to execute kernel mode instructions, the system must switch from user mode to kernel mode and starts running code from the kernel in order to answer the system call



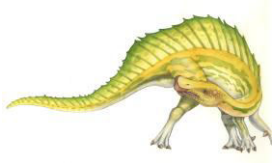
1. What is “user space”?
2. What is “kernel space”?
3. Where is a user process, user space or the kernel space?
4. Where is a thread, in user space or kernel space?



System calls

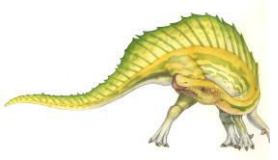
- **System calls** provide an interface to the services made available by an operating system
- There are many system calls, one particular Linux kernel has 393 different system calls:
- getitimer(), getpageside(), getpid(), fork(), open(), close(), read(), reboot(), getcpu(), write(), clone()

1. Why pthread_create() is not trapped as a system call?
2. When a thread make a system call (and transit to kernel mode), does the thread still run in user space?
3. While running in kernel mode, does a user thread modifies its thread stack in the process (each user thread as a stack in the process that has created it)?



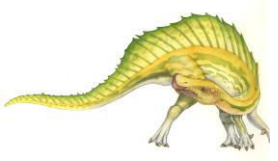
Processes & threads

1. Does two processes share anything?
2. Does clone() always create a new task?
3. Does tasks share anything?
4. What does threads share with processes and with each other?
5. In Linux, what is PID and TID?
6. Are PID and TID assigned by user process or OS kernel?
7. A thread is
    1. A TCB execution context (PC,registers, stack pointers, etc)
    2. Or a sequence of instructions been executed
    3. Or both 1 and 2
8. If a thread is a TCB, does all threads are "kernel threads"?
9. Can the PC be modified from inside a user process?



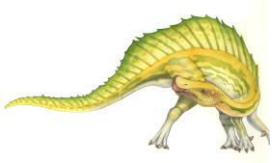
User/kernel threads and stacks

1. What is the relation between stack and thread?
2. Is there a stack for each thread?
3. What is the difference between kernel stack and user stack?
4. Does threads always have a kernel stack?
5. Can a thread use its user stack to run in kernel mode?
6. Does the kernel know about the user thread and the user stack?
7. Does kernel create its own kernel threads or does there is only kernel threads mapping with user threads?
8. What is the key difference between user thread and kernel thread?
9. Does all threads belong to a same process?



Threads scheduling

- Two water pipes but one faucet
- Only one water flow can go through the faucet
- The water pipes are the threads
- The faucet is the CPU
- The CPU can process instructions of only one thread at any given CPU cycle
- Threads scheduling is to decide from which thread a sequence of instructions is executed at any given time



Challenges in scheduling threads

- Some of the issues scheduling algorithms face:
  1. There is usually a very large number of threads
  2. Threads belong to different processes
  3. Threads have different priorities
  4. Threads may have to be executed before others
  5. Threads may be blocked, thus cannot be scheduled