OPERATING SYSTEM

Operating System Threads

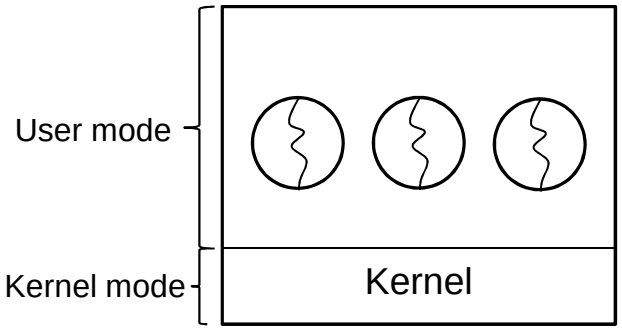
Introduction to threads

Types of threads

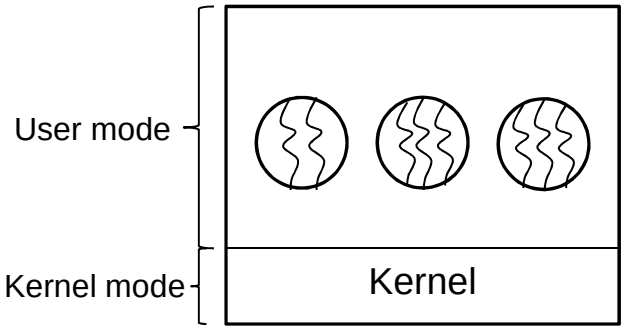
Multi-threading models

Introduction

- Process is a huge code that gain access to microprocessor/resources to perform particular task
- Term Process has provided many facilities to operating system but still it looks like a huge block of code to be managed
- Solution is to use a process into multiple small pieces (called threads)
- Threads are small executable piece of process those can be handled as a task
- Process(es) within a process
- By doing so, it is easy to manage threads rather than managing giant process



Single Thread Process



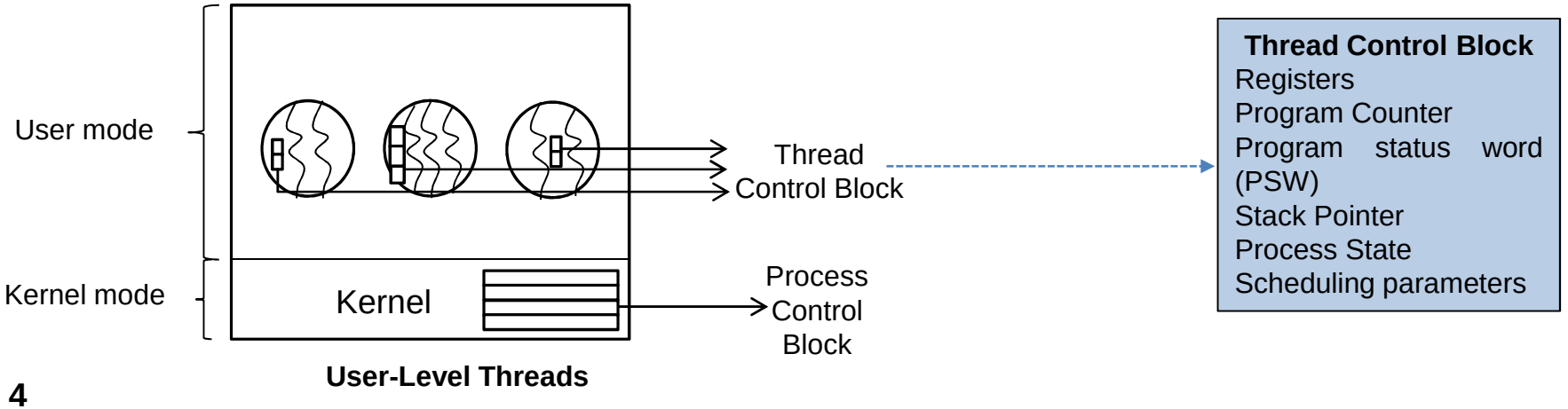
Multiple Thread Processes

Advantages

- Increase User Responsiveness
- Threads of same process can share resource(s) with each other
- Management of threads take less time then process (Threads are lighter & faster then process)
- Multi-threading increases Microprocessor Utilization, hence increase in performance
- Threads can be a factor of performance if all of threads (of one process) are not CPU-Bound only

User-Level Threads

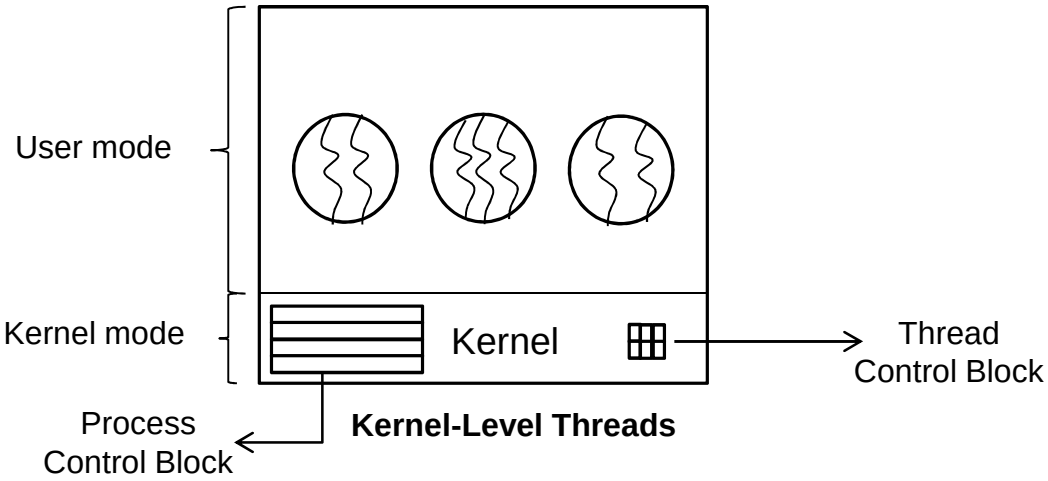
- Threads are created at user level
- Application programs are responsible to create multiple threads
- These threads can be created by a programming languages support (e.g. Java Virtual Machine), Libraries (e.g. OpenMP) or by APIs (e.g. POSIX)
- Disadvantages:
  - If a thread is blocked from parent process, it should be block from child too…!
  - If a thread was using a resource & new thread was assigned and wants the same resource, then how resource should be managed…!



User-Level Threads

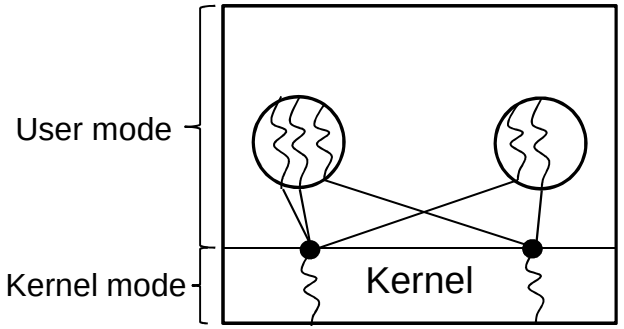
Kernel-Level Threads

- Threads are created at kernel level
- Operating System or kernel is responsible to break a process into multiple threads
- TCB is managed by kernel so it is easy to overcome on previous disadvantages
- Disadvantages:
  - Load over kernel
  - When processes communication is performed, then threads will interact on those communications/signals



Hybrid Threads

- Combination of both
- Also known as LWP (Light Weight Process)
- User-level threads are multiplexed into kernel-level thread
- Each kernel-level thread is invoked by a set of user-level thread (depends on the nature of call)
- In this case, kernel only manages kernel-level threads
- While processes threads are managed as user-level threads



Hybrid Threading

Multithreading Models

- Many systems provide support for both user-level and kernel level thread which provides different multithreading models:
  - Many-to-One (User-level threads): maps many user level threads into one kernel level thread.
    - The entire process will block if a thread makes a blocking system call
    - Since only one thread can access kernel at a time, multiple threads cannot run concurrently and thus cannot make use of multiprocessors
  - One-to-One (Kernel-level threads): maps each user level thread to a kernel level thread
    - Creating a user level thread results in creating a kernel thread
    - More overhead and allows parallelism
    - Because of the overhead most implementations limit the number of kernel threads created
  - Many-to-Many (Hybrid): Multiplexes many user level threads to a smaller or equal number of kernel threads
    - Has the advantages of both the many-to-one and one-to-one model

Questions