OPERATING SYSTEMS
CSE-4041

Lecture-4

Multithreading

Nitesh Kumar Jha

Assistant Professor

ITER S'O'A (Deemed to be University)

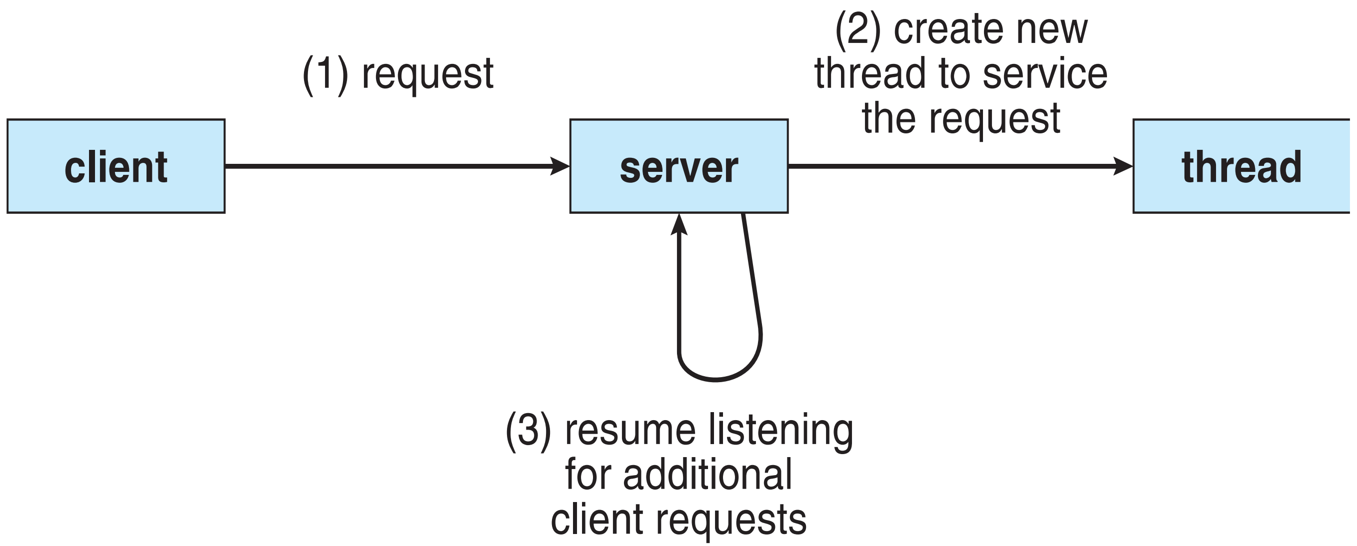
Threads

- A thread is a basic unit of cpu utilization.
- It comprises of a thread ID, a program counter, a register set and a stack.
- A traditional process has a single thread of control. If a process has multiple thread of control. If a process has multiple thread of control it can perform more than one task at a time.

Multithreading

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

Multithreaded Server Architecture



Benefits

- **Responsiveness** – May allow continued execution if part of process is blocked, especially important for user interfaces.
  For instance a web browser could allow user interaction in one thread while an image was loaded in another thread.
- **Resource Sharing** – Threads share resources of process, easier than shared memory or message passing. *Benefit of sharing code and data allows application to have several threads of activity within the same address space.*
- **Economy** – Cheaper than process creation, thread switching lower overhead than context switching. In general it is more time consuming to create and manage process than threads.
- **Scalability** – Process can take advantage of multiprocessor architectures. Single threaded process can run on one processor. Multi threading on a multi cpu machine increases parallelism.
- *For Ex. A word processor(e.g., MS word) may have a thread for displaying graphics, another thread for keystrokes, another thread for spell checking and so on.*

Multicore Programming

- Multicore or multiprocessor systems putting pressure on programmers, challenges include:
  - **Dividing activities:** find areas that can be divided into separate concurrent task.
  - **Balance:** balance of task equal work as task run in parallel
  - **Data splitting:** Data accessed and manipulated must be divided to run in separate areas.
  - **Data dependency:** one task depends on other tasks data therefore execution of task must be synchronized.
  - **Testing and debugging:** Different testing and debugging task for programs must run in parallel on multiple cores
- ***Parallelism*** implies a system can perform more than one task simultaneously
- ***Concurrency*** supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

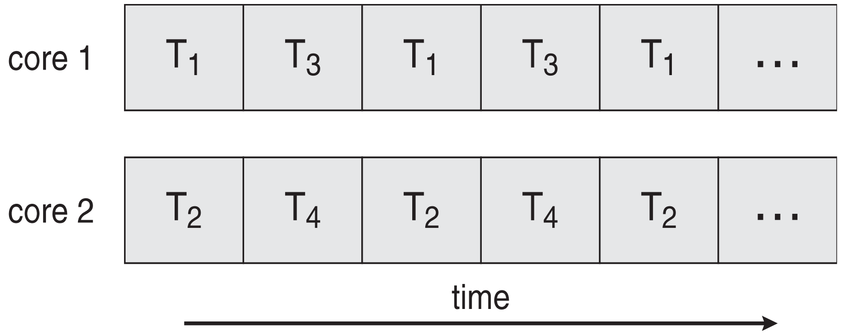
Multicore Programming (Cont.)

- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
  - CPUs have cores as well as ***hardware threads***
  - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core

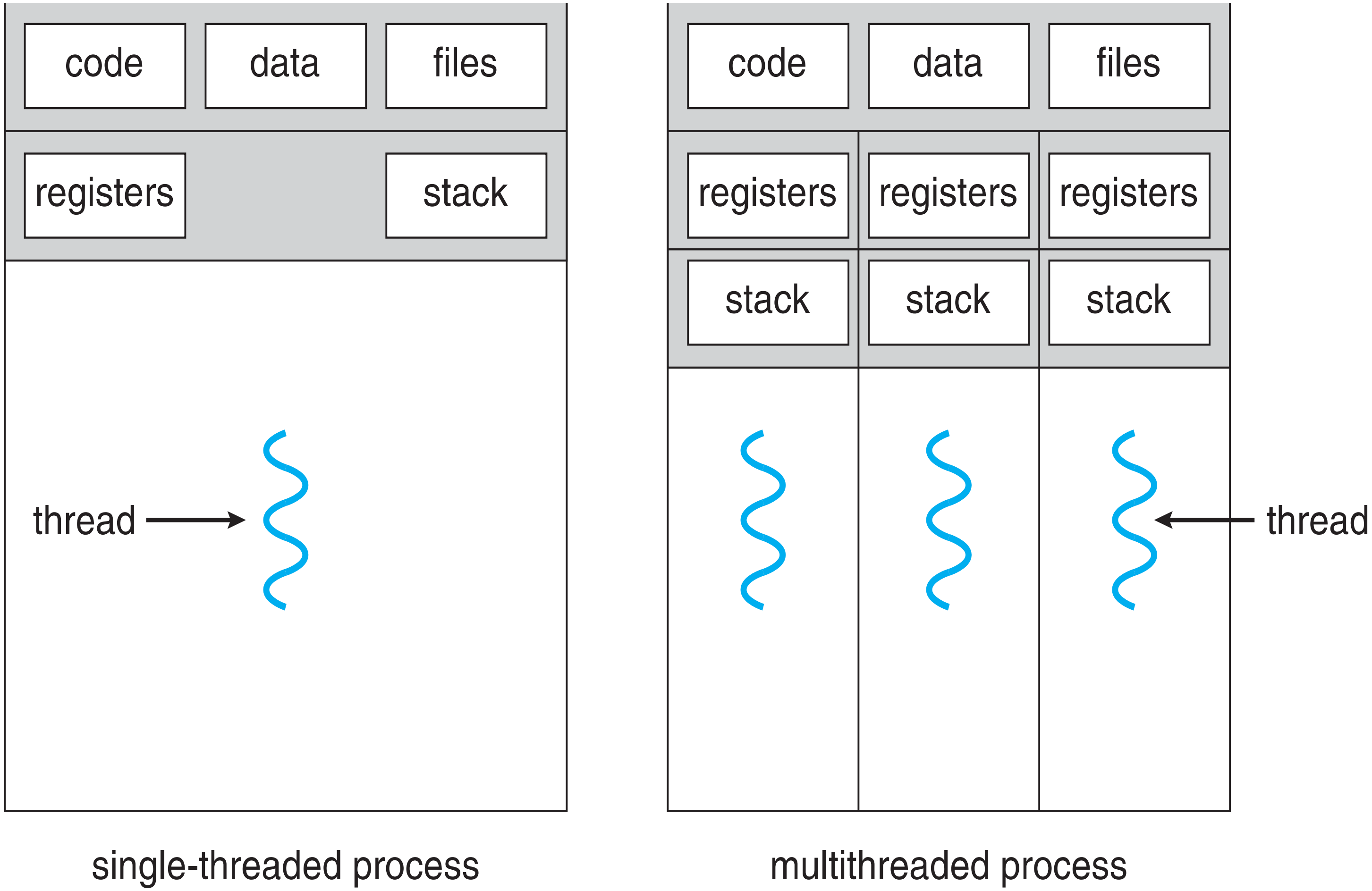
Concurrency vs. Parallelism

- Concurrent execution on single-core system:

- Parallelism on a multi-core system:



Single and Multithreaded Processes



Amdahl's Law

- Amdahl's Law is a formulation, an observation that attempts to give an upper bound on the performance increase that can be realized when a fixed workload is moved to a more parallel environment (e.g. multiprocessing or multi-core environment).
- *S* is serial portion
- *N* processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- As *N* approaches infinity, speedup approaches 1 / *S*

Serial portion of an application has disproportionate effect on performance gained by adding additional cores

Amdahl's Law

- You can also consider speedup $\leq \dfrac{1}{(1-p+\frac{p}{N})}$ where p = parallel portion and N = processing cores

- If 30% of the execution time may be the subject of a speedup, *S* will be 0.7 and p=0.3; if the improvement makes the affected part twice as fast, *N* will be 2. Amdahl's law states that the overall speedup of applying the improvement will be????
- Assume that we are given a serial task which is split into four consecutive parts, whose percentages of execution time are *p1* = 0.11, *p2* = 0.18, *p3* = 0.23, and *p4* = 0.48 respectively. Then we are told that the 1st part is not sped up, so *N1* = 1, while the 2nd part is sped up 5 times, so *N2* = 5, the 3rd part is sped up 20 times, so *N3* = 20, and the 4th part is sped up 1.6 times, so *N4* = 1.6. By using Amdahl's law, Find the overall speedup ????

User Threads and Kernel Threads

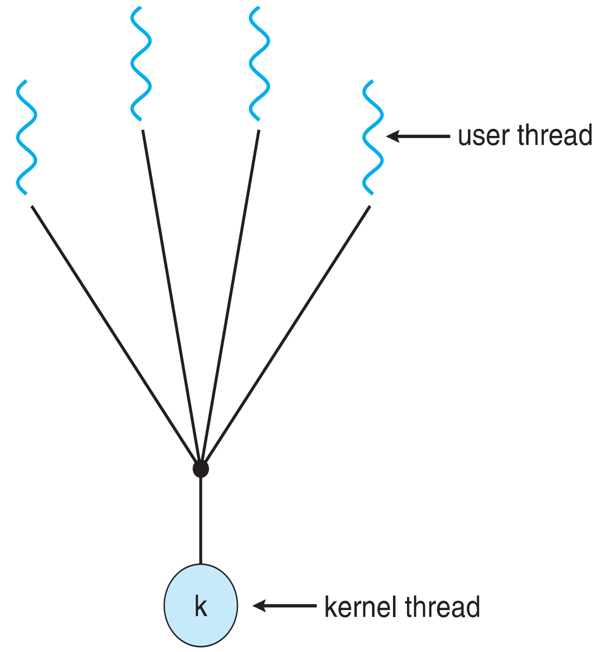
- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

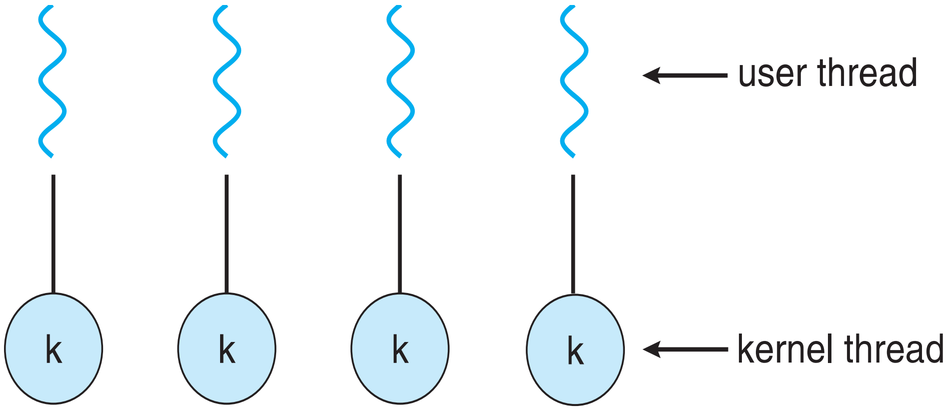
Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



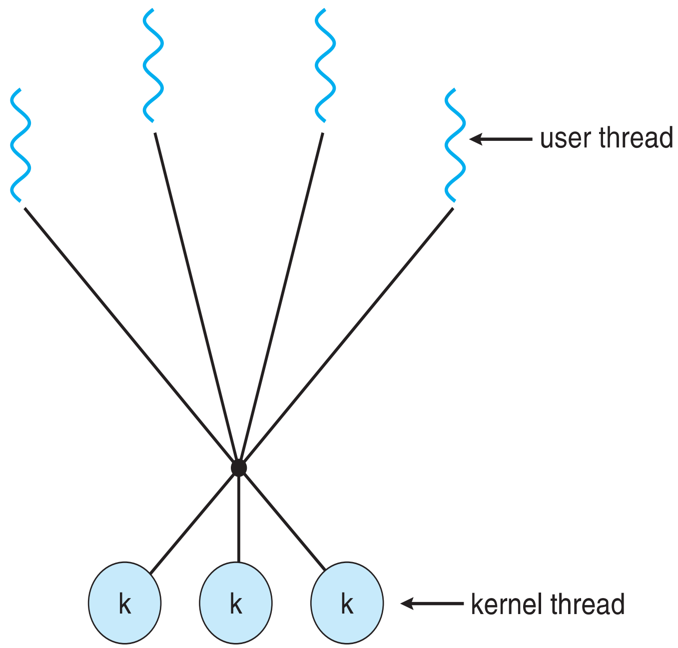
One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



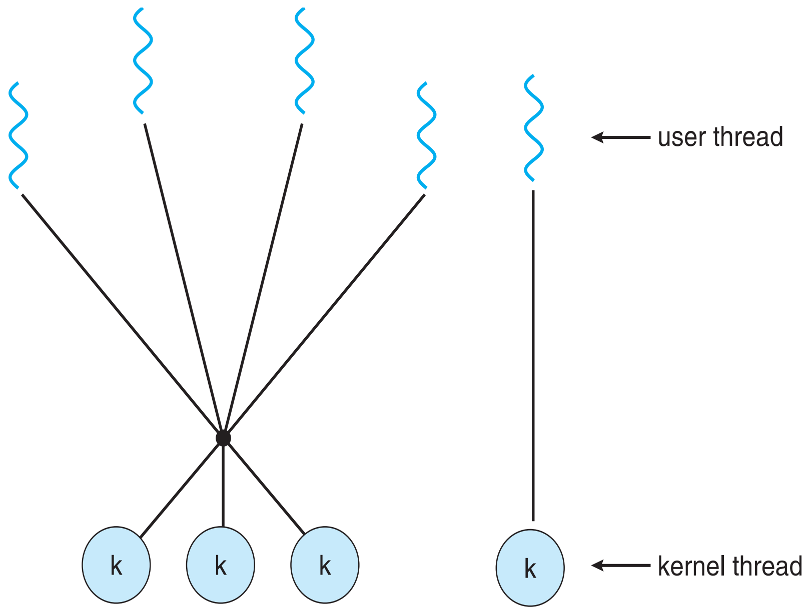
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *ThreadFiber* package



Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



End of Lecture

Thank You