PES UNIVERSITY ONLINE

OPERATING SYSTEMS

Threads and Concurrency

Chandravva Hebbi
Department of Computer Science

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Threads and Concurrency

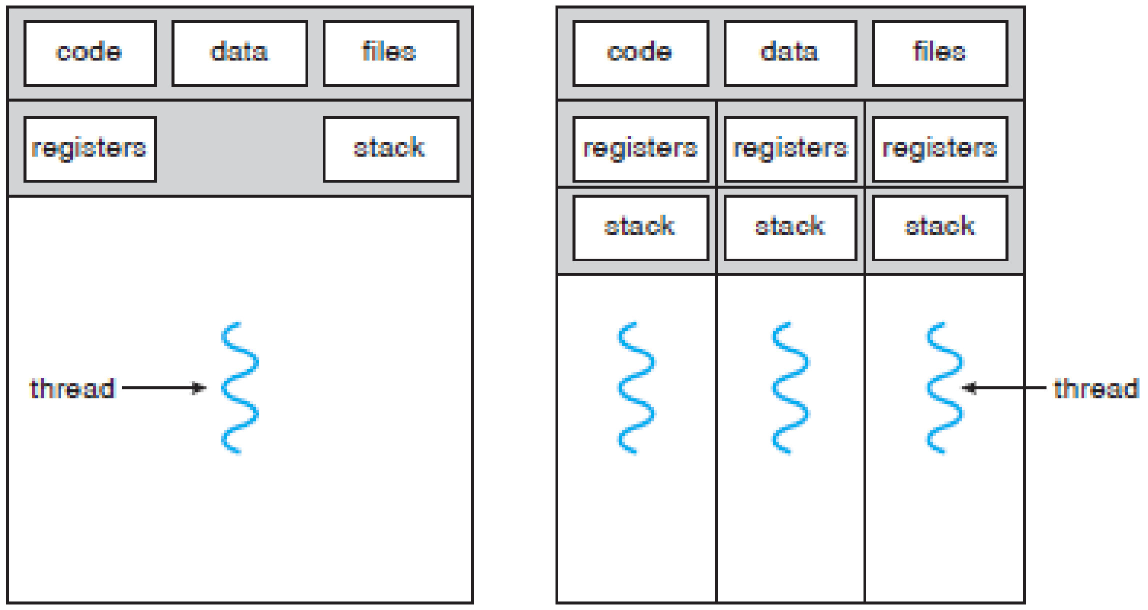
Chandravva Hebbi
Department of Computer Science

Overview and Motivation

- A Thread is a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems.
- It consists of thread ID, Program counter, a register set and stack
- Shares with other threads of same process its code, data,file descriptors, signals
- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads.
- Application 1: internet browser.
  - numerous tabs open at a given time
  - Multiple threads of execution are used to load content, display animations, play a video, fetch data from a network and so on.

Single threaded and multithreaded processes

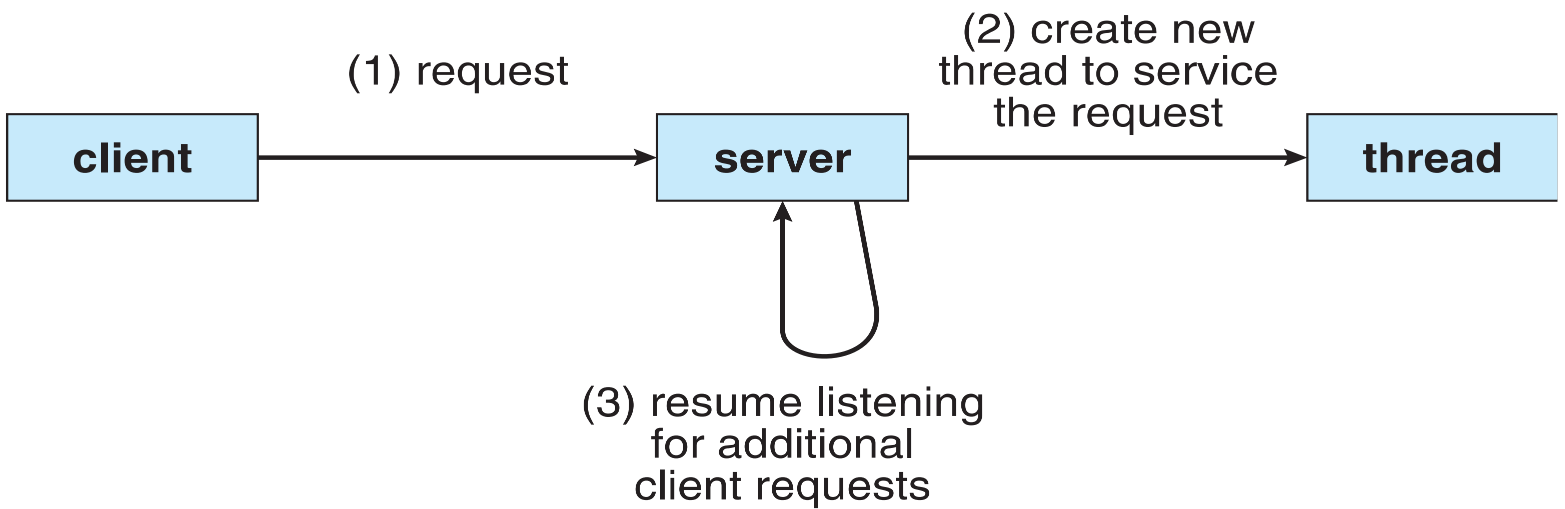
- Application 2: Word processor
  - Thread to accept input
  - A thread for spell checking
  - A thread for grammar checking



single-threaded process   multithreaded process

Multithreaded Server Architecture

- Process creation is heavy-weight while thread creation is light-weight
- Process creation if time consuming, resource intensive
- Threads also play a vital role in remote procedure call (RPC) systems
- Can simplify code, increase efficiency
- Kernels are generally multithreaded


(1) request
client
server
(2) create new thread to service the request
thread
(3) resume listening for additional client requests

OPERATING SYSTEMS

Benefits

- ~~**Responsiveness** – may allow continued execution if part of~~ process is blocked or performing lengthy operations.
  - It is useful in designing user interfaces.
  - A multithreaded Web browser could allow user interaction in one thread while an image was being loaded in another thread.
  - Example: click on the button
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing.
  - Programmer needs to specify the techniques for sharing
  - But threads share the memory and other resources
  - Sharing of resources helps in having many threads within the same address space

Benefits

- **Economy –** cheaper than process creation, thread switching lower overhead than context switching.
    - In Solaris, for example, creating a process is about thirty times slower than creating a thread, and context switching is about five times slower.
- **Scalability –** process can take advantage of multiprocessor architectures.
    - Threads can run on multiple cores parallelly

Processes vs Threads

Process	Thread
• Will by default not share memory	• Will by default share memory
• Most file descriptors not shared	• Will share file descriptors
• Don't share filesystem context	• Will share filesystem context
• Don't share signal handling	• Will share signal handling

Attributes shared by Threads

- process ID and parent process ID;process group ID and session ID;
- controlling terminal;
- process credentials (user and group Ids);open file descriptors;
- record locks created using fcntl();signal dispositions;
- file system–related information: umask, cwd and root directory;
- resource limits;CPU time consumed (as returned by times());
- resources consumed (as returned by getrusage()); nice value (set by setpriority() and nice()).

Attributes specific to Threads

- thread ID ;signal mask;
- Thread-specific data ;
- the errno variable;
- floating-point environment (see fenv(3));
- stack (local variables and function call linkage information i.e CPU registers saved in the called function's stack frame when one function calls another function and restored for the calling function when the called function returns)
- and a few more …….

Thread Concepts

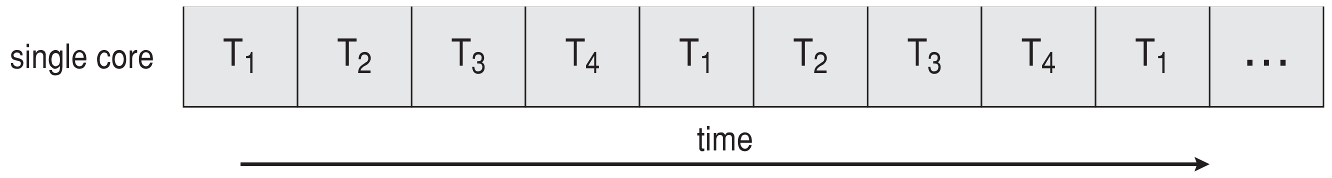
- A thread consists of the following information necessary to represent an execution context within a process.
- thread ID that identifies the thread within a process
- Every thread has a thread ID
- set of register values
- stack
- scheduling priority and policy
- signal mask
- Errno variable
- Thread-specific/thread-private data (each thread to access its own separate copy of the data, without worrying about synchronizing access with other threads)

Thread Concepts (Cont.)

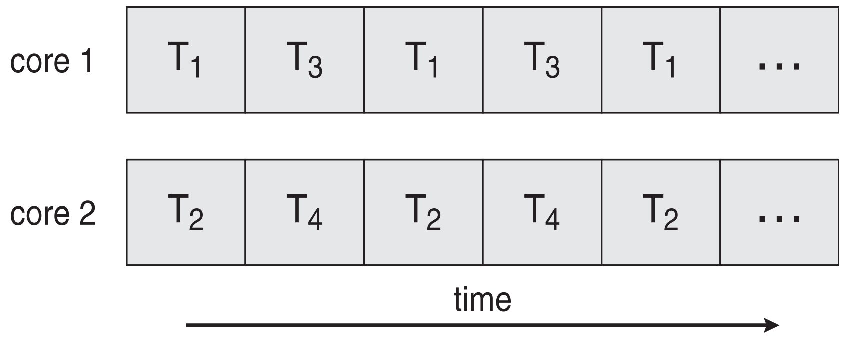
- When a thread is created, there is no guarantee which will run first: the newly created thread or the calling thread.
- The newly created thread has access to the process address space and inherits the calling thread’s floating-point environment and signal mask
- The set of pending signals for the thread is cleared.
- The pthread functions usually return an error code when they fail. They don’t set errno like the other POSIX functions.

Concurrency vs. Parallelism

- Concurrent execution on single-core system:



- Parallelism on a multi-core system:



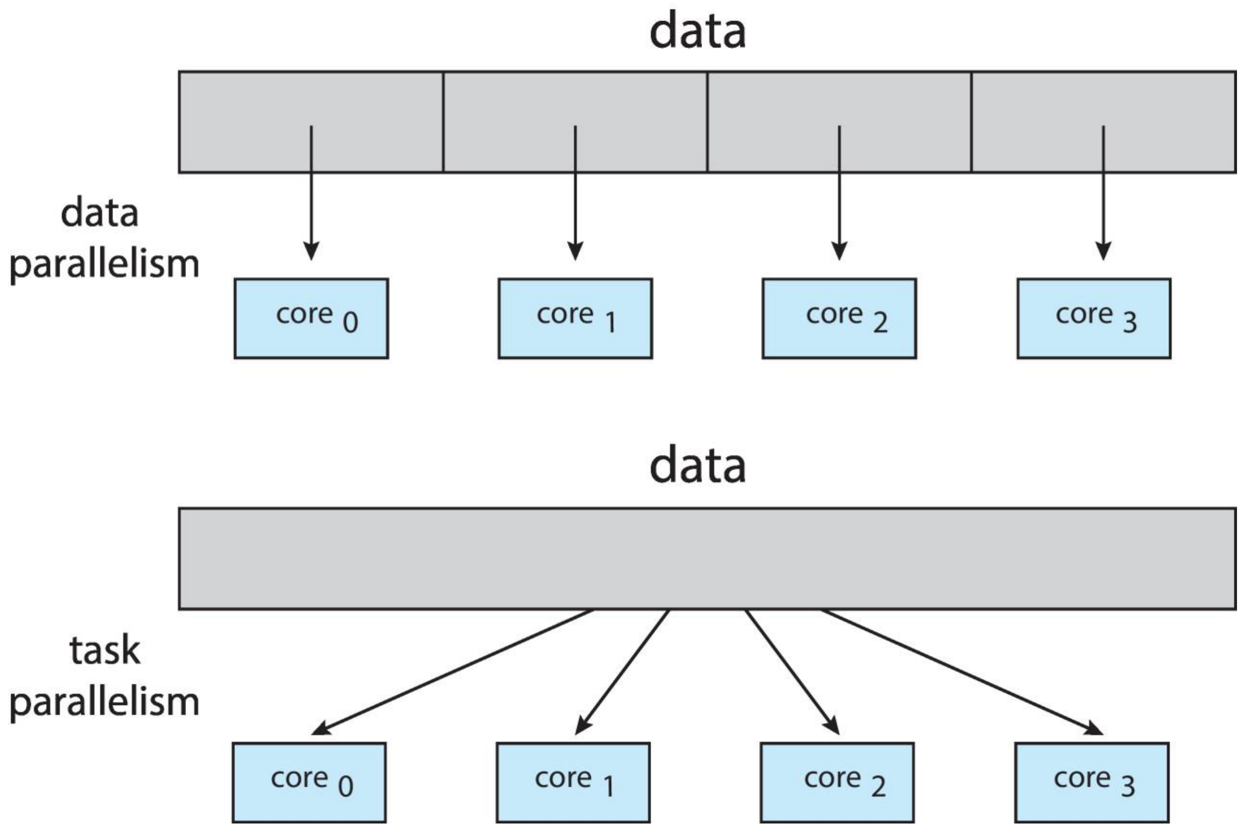
Multicore Programming

- Multicore or multiprocessor systems putting pressure on programmers, challenges include:
  - **Dividing activities**
  - **Balance**
  - **Data splitting**
  - **Data dependency**
  - **Testing and debugging**
- ***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

OPERATING SYSTEMS
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

Data and Task Parallelism



Amdahl's Law

- Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- *S* is portion of an application that needs to be done in serial
- *N* processing cores

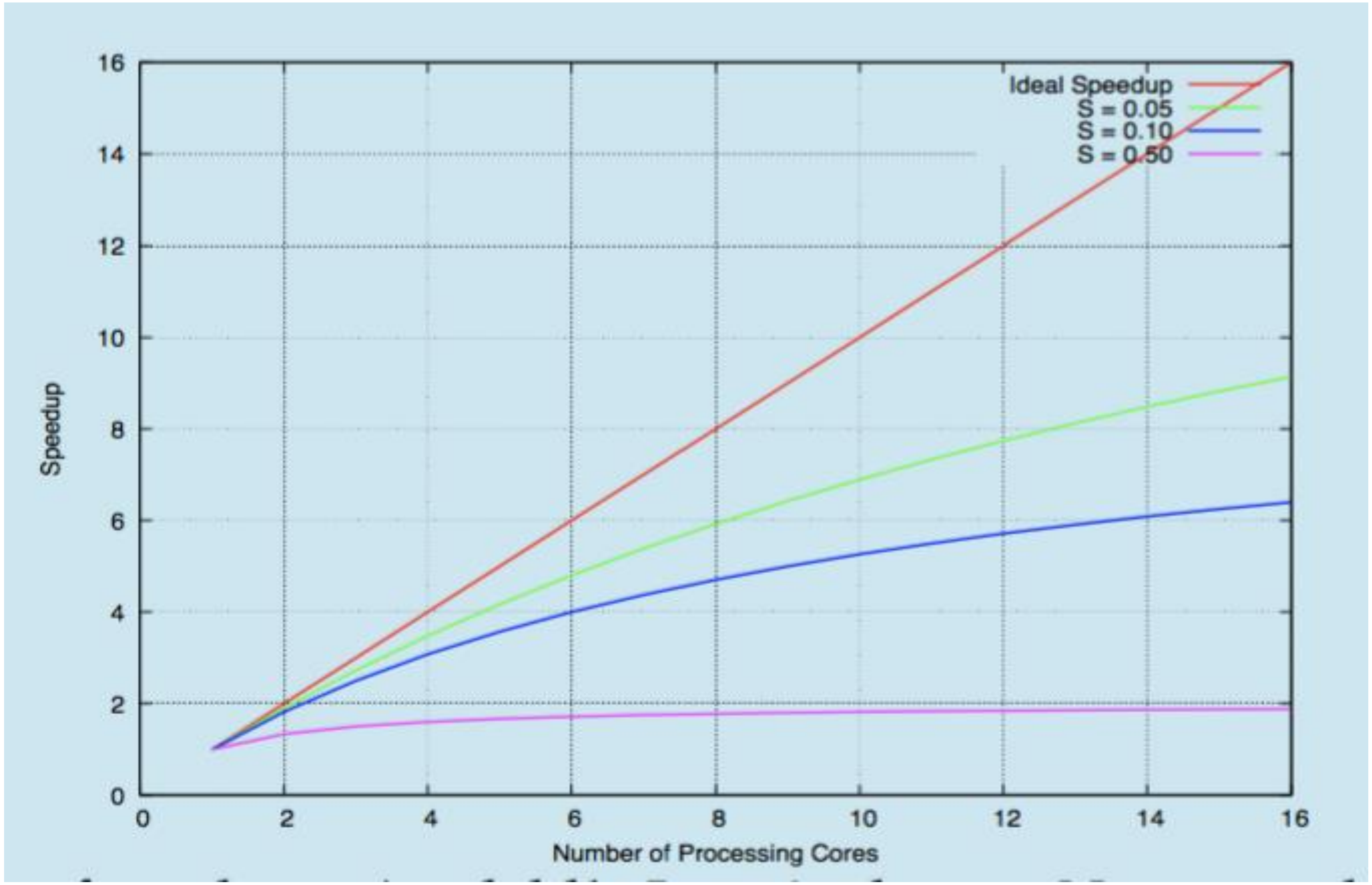
$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- 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
- But does the law take into account contemporary multicore systems?

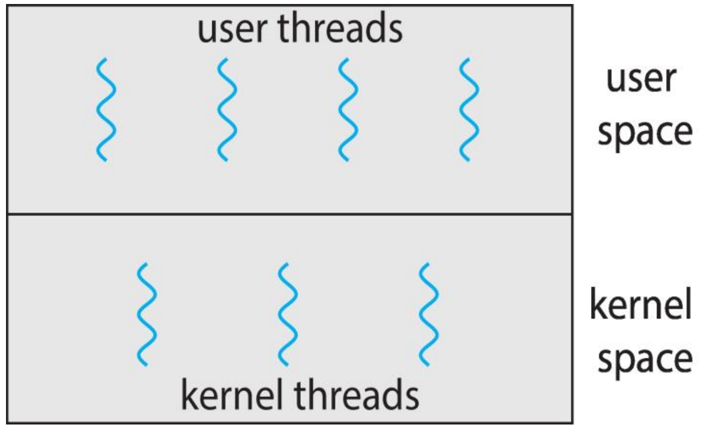
OPERATING SYSTEMS

Amdahl's Law



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



THANK YOU

Chandravva Hebbi
Department of Computer Science Engineering
chandravvahebbi@pes.edu