

# Processes

- One of the most important concepts in all modern operating systems
- A process is a **container** (or execution environment) for a program (software) in execution.
- Any software is executed with one or more processes.
  - Java VM (JVM), MS Word, Excel, PPT, Firefox, iTunes, Google Chrome, etc.

## Concurrency with Threads

1

2

## Multi-Tasking (Time Sharing)

- All modern OSes support **multi-tasking**.
  - Doing **multiple things** with **multiple processes** on a single **CPU** (more precisely, a single CPU core)
    - e.g., writing a document with MS Word while playing a music with iTunes and downloading files with a Web browser
- At any moment, a single CPU core can execute a single program (process).
- **Pseudo concurrency/parallelism**: An **illusion** for human eyes as if multiple processes are executed at the same time.
  - OS periodically assigns one process to another to the CPU core.
    - every several tens of milliseconds to several hundreds of milliseconds
    - A CPU core is multiplexed among processes; processes are not executed in a completely parallel manner.

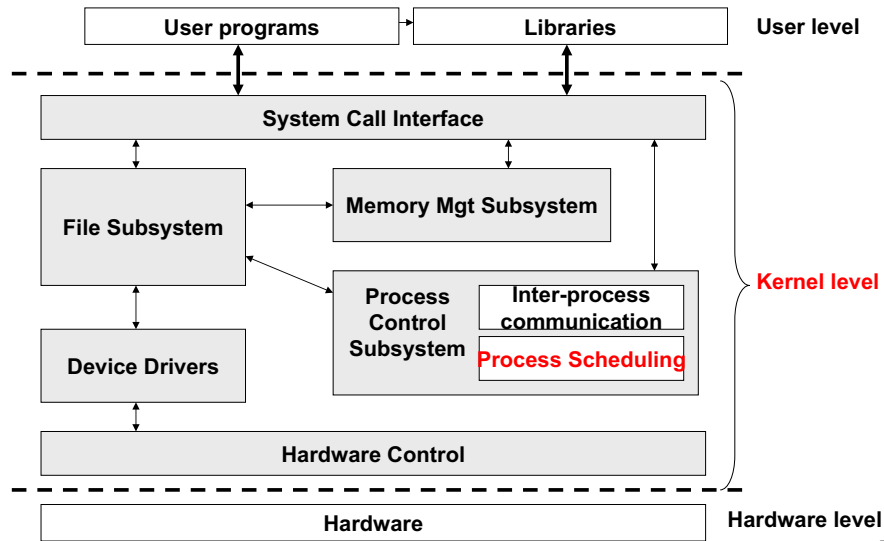
3

## Example Processes

PID	Process Name	User	% CPU	Threads	Real Mem	Kind
11480	PepperFlashPlayer (Chrome Plug-In Host)	jks	20.7	167	711.9 MB	Intel
339	Thunderbird	jks	5.4	32	582.9 MB	Intel (64 bit)
9302	iTunes	jks	0.2	32	513.7 MB	Intel (64 bit)
11464	Google Chrome	jks	1.3	34	252.8 MB	Intel (64 bit)
11862	Google Chrome Renderer	jks	0.1	5	199.3 MB	Intel
14884	Google Chrome Renderer	jks	0.2	8	146.9 MB	Intel
7799	Finder	jks	0.5	14	146.7 MB	Intel (64 bit)
16313	HandBrake	jks	153.8	25	142.9 MB	Intel (64 bit)
15525	Google Chrome Renderer	jks	0.0	5	89.1 MB	Intel
15540	Google Chrome Renderer	jks	1.8	5	86.3 MB	Intel
15450	Screen Sharing	jks	18.4	5	83.5 MB	Intel (64 bit)
11497	Google Chrome Renderer	jks	0.2	5	83.1 MB	Intel
14413	Google Chrome Renderer	jks	0.1	5	80.6 MB	Intel
14423	Google Chrome Renderer	jks	0.4	5	80.3 MB	Intel
14414	Google Chrome Renderer	jks	0.1	6	80.3 MB	Intel
14426	Google Chrome Renderer	jks	0.1	5	79.9 MB	Intel
16212	Google Chrome Renderer	jks	0.2	5	79.7 MB	Intel
14425	Google Chrome Renderer	jks	0.1	5	79.4 MB	Intel

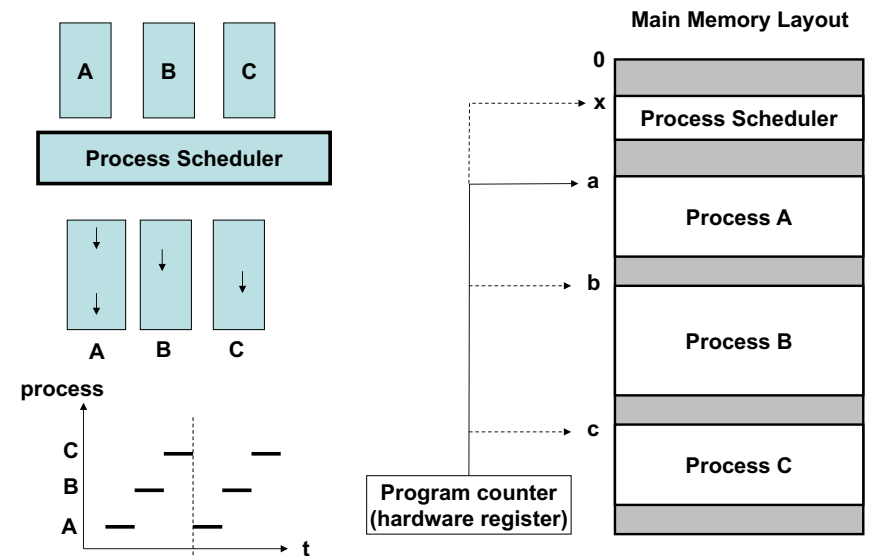
4

# An Architectural View of an OS



5

# Process Scheduling



6

## Why Threads?

- **Good old days:** Process-based, **coarse-grained** concurrency was enough
  - e.g., Editing a document while downloading files
- **Now:** **Fine-grained** concurrency is required.
  - Each program is required to execute **in-program concurrent tasks**.
    - Doing different things at the same time.
    - Doing the same or a similar thing for multiple clients at the same time.

7

- Each program is required to do different things at the same time.
  - **Word**
    - Displaying text and images
    - Responding to keystrokes and mouse inputs from the user
    - Downloading fancy document templates from MS web site
    - Performing spelling and grammar checking in the background
  - **Web browser**
    - Displaying text and images
    - Responding to keystrokes/mouse inputs
    - Checking and downloading software updates
    - Multiple tabs
  - **iTunes**
    - Playing music
    - Downloading music and its metadata (album's cover img, song titles, lyrics...)

8

- # of threads
  - Kernel tasks: 121
  - Dropbox: 103
  - Firefox: 74
  - Thunderbird: 57
  - iTunes: 30
  - MS PPT: 21

– Each program is required to do the same or a similar thing for multiple clients at the same time.

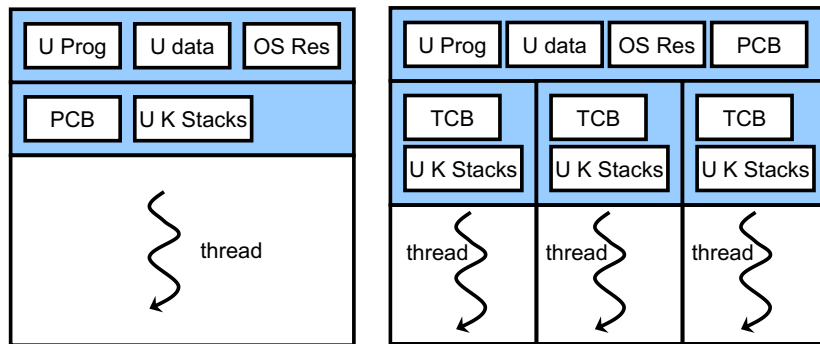
- Web server
  - Accepts and parses an HTTP request
  - Finds a target file
  - Makes an HTTP message (header, payload, etc.)
  - Returns a target file with the HTTP message

- Assign **threads** to in-program concurrent tasks

9

10

- Process-creation is heavyweight.
  - Time-consuming and resource intensive.
    - Creating a process is 30 times slower than creating a thread.
    - Process switching is 5 times slower than thread switching.



Single-threaded (traditional) process

Multi-threaded process

11

- **Fine-grained pseudo parallelism**: An illusion for human eyes as if multiple threads are executed on a process at the same time.
  - OS periodically assigns one thread to another to the CPU core.
    - A CPU core is multiplexed among processes and threads; processes/threads are not executed in a completely parallel manner on a single CPU core.

12

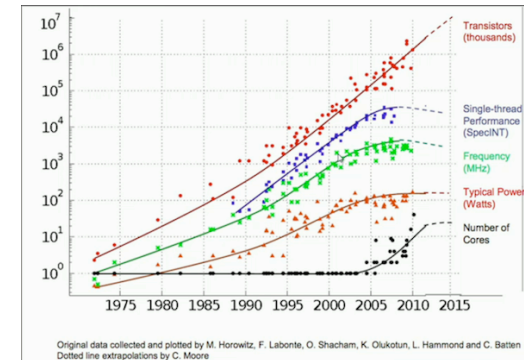
## Summary: Why Threads?

- Responsiveness/availability
  - Threads allow a program to continue running even if a part of it is blocked for I/O or is performing a long operation.
- Resource sharing
  - Threads share the memory and other resources of a process that they belong to.
- Efficiency
  - Threads are more lightweight than processes.
    - Process creation is expensive.
    - Switching processes is heavyweight.

13

## Other Viewpoints to Threads

- CPU speed does not increase any more as it did in the past...
  - Physical material barrier to increase clock speed and # of transistors in a CPU
  - Heat (and cooling) and power consumption problems



14

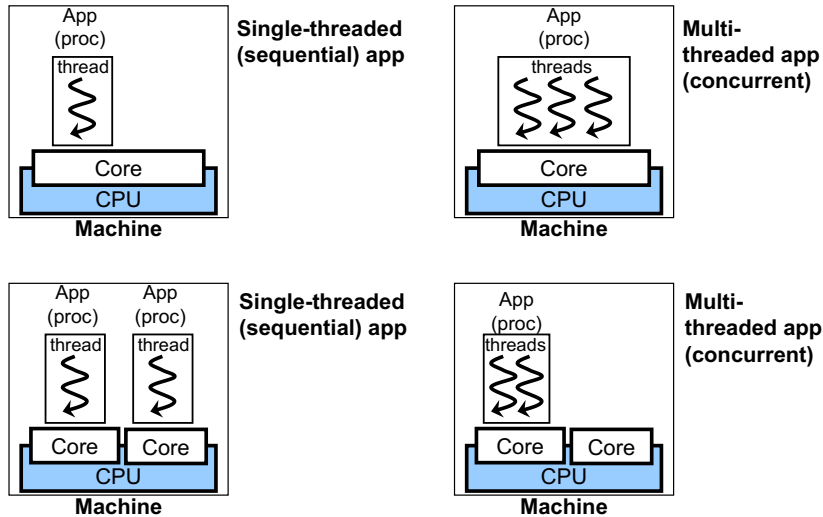
- Each CPU increases its density of cores, not its speed.
  - Intel Core i7: up to 4.0 GHz and up to 8 cores
  - Intel Xeon: up to 4.0 GHz and up to 8 cores
  - Multiple relatively-slower CPU cores for mobile devices to reduce power consumption
    - Amazon Kindle Fire (\$50): Quad-core 1.3 GHz
    - Apple iPhone 8/10: 6 cores
- Significant energy consumption and CO<sub>2</sub> emission by data centers
  - Amazon (450K+ servers), Facebook, Google (500K+ servers), Microsoft, Salesforce.com and Yahoo!
  - DCs consumed approx. 1.5% of global electricity usage ('10).
  - Google paid \$2M/mo for electricity bills ('07).
  - The Info. and Comm. Technology (ICT) industry produced 2% of global CO<sub>2</sub> emission ('07).
    - On par with the aviation industry.
  - IDCs were responsible for 23% of the ICT industry's emission

15

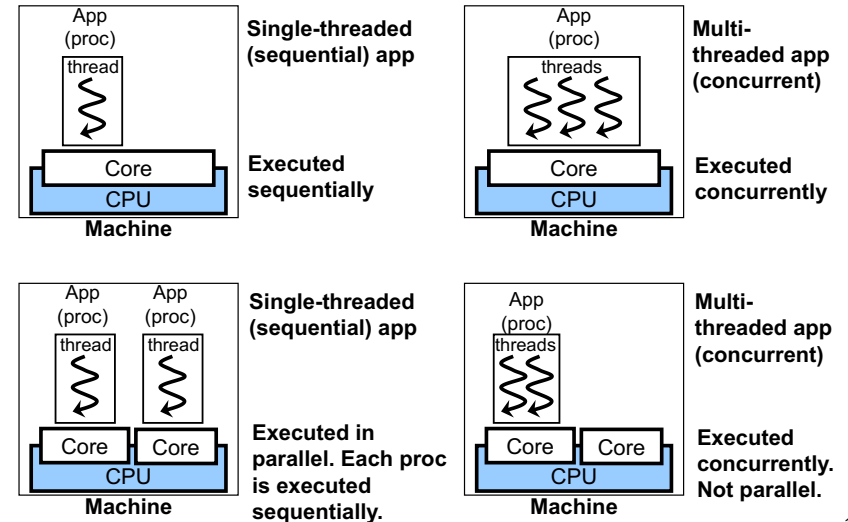
- GPUs are designed with a massive # of cores.
  - NVIDIA Tesla K80: 4992 cores
- You are expected to increase your app's performance by increasing its concurrency.
- Ultimate goals of multi-threaded applications:  
**Responsiveness and performance improvement**

16

## Terminology: Single- or Multi-threaded?

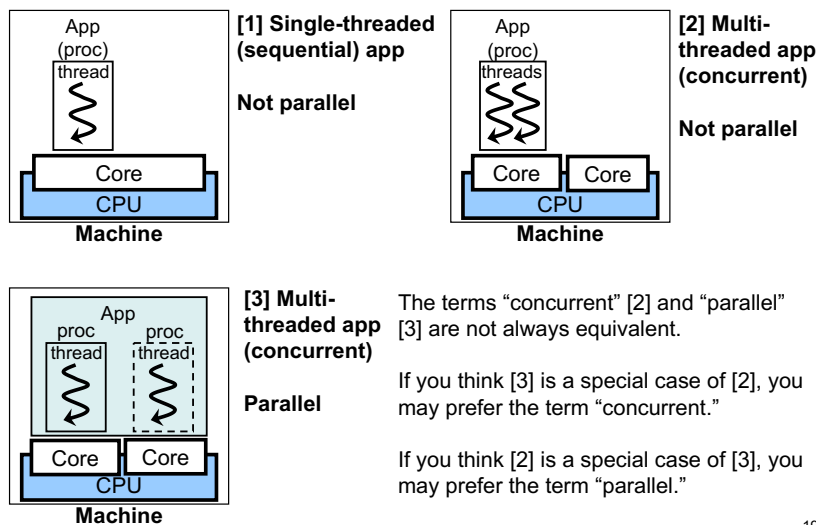


17



18

## Terminology: Concurrent or Parallel?



19

## In Java...

- The terms "concurrent" and "parallel" are somewhat mixed up in Java.
  - Java 5 (2004) introduced `java.util.concurrent`, which was further enhanced by Java 6 (2006).
    - "Concurrent" collections such as `ConcurrentHashMap`
    - The Executor framework
      - An extension/abstraction over low-level threads
  - Java 7 (2011)
    - The Fork/Join framework, an extension to the Executor framework
      - The term "parallel" appeared in its documentation, although it was placed in `java.util.concurrent`.
    - "Parallel" garbage collector
  - Java 8 (2014)
    - "Parallel" collection streams, which allows for "parallel" (threaded) operations on collections with lambda expressions
      - e.g. MapReduce-inspired API extension for collections

20

- An example of using a parallel stream.

```
- Integer price = cars.stream()
    .parallel()
    .map((Car car)-> car.getPrice())
    .reduce(0, (result, carPrice)->{
        if(result==0) return carPrice;
        else if(carPrice < result) return carPrice;
        else return result;} );
```

## Java Threads

21

## Java Threads

- Every Java program has at least one *thread of control*.
  - main() runs with a thread on a JVM.
    - The “main” thread is *implicitly* created when a JVM starts.
- If you need extra threads in addition to the main thread, you need to *explicitly* create them.
- 4 things to do:
  - Define a class implementing the `java.lang.Runnable` interface
    - `public abstract void run();`
  - Write a threaded/concurrent task in `run()` in the class
  - Instantiate `java.lang.Thread` and associate a `Runnable` object with the thread
  - Start (call `start()` on) the instantiated thread.
    - `run()` is automatically called on the thread.

23

22

## An Example Code: Creating a Thread

- GreetingRunnable.java

```
- class GreetingRunnable implements Runnable{
    private String greeting;

    public GreetingRunnable(String aGreeting){
        greeting = aGreeting; }

    public void run(){
        for( int i=0; i<10; i++ ){
            Date now = new Date();
            System.out.println(now + " " + greeting);
        } }
```

- HelloWorldTest.java

```
- main(...){
    GreetingRunnable runnable1 = new GreetingRunnable("Hello World");
    Thread thread1 = new Thread(runnable1);
    thread1.start(); }
```

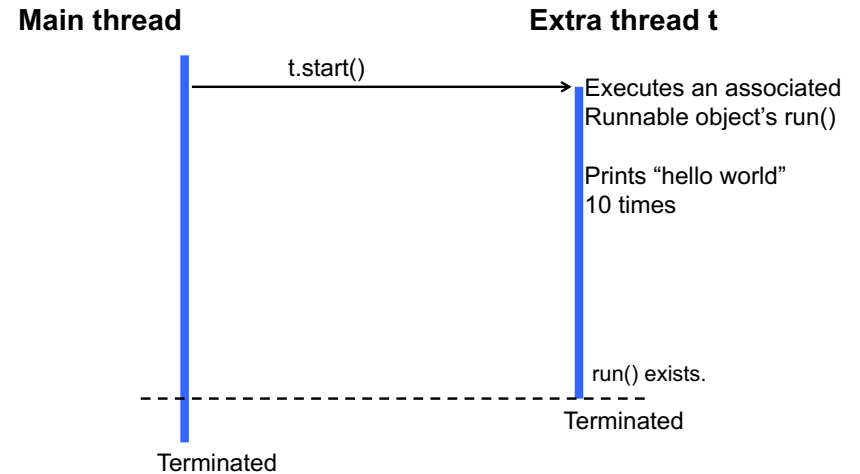
24

## Thread.start()

- Creating a Thread object does not mean creating a new thread.
  - It is start() that actually creates a thread.
- start()
  - Allocates memory and initializes a new thread on a JVM.
  - Calls run() of a specified Runnable object.
    - Do not call run() directly, but let start() call run() on behalf of yourself.

25

## Program Execution



26

## An Example Code: Creating Threads

- Output:
  - Mon Mar 26 15:14:43 EDT 2007 Hello World
  - Mon Mar 26 15:14:44 EDT 2007 Hello World
  - Mon Mar 26 15:14:45 EDT 2007 Hello World
  - Mon Mar 26 15:14:46 EDT 2007 Hello World
  - Mon Mar 26 15:14:47 EDT 2007 Hello World
  - Mon Mar 26 15:14:48 EDT 2007 Hello World
  - Mon Mar 26 15:14:49 EDT 2007 Hello World
  - Mon Mar 26 15:14:50 EDT 2007 Hello World
  - Mon Mar 26 15:14:51 EDT 2007 Hello World
  - Mon Mar 26 15:14:52 EDT 2007 Hello World

27

- HelloWorldTest2.java and GreetingRunnable.java

```
main(...){
    GreetingRunnable runnable1 = new GreetingRunnable("Hello World");
    GreetingRunnable runnable2 = new GreetingRunnable("Goodbye World");

    Thread thread1 = new Thread(runnable1);
    Thread thread2 = new Thread(runnable2);

    thread1.start();
    thread2.start();
}
```

28

## The Order of Thread Execution

- Output:

```
- Mon Mar 26 15:28:45 EDT 2007 Goodbye World
- Mon Mar 26 15:28:45 EDT 2007 Hello World
- Mon Mar 26 15:28:46 EDT 2007 Hello World
- Mon Mar 26 15:28:46 EDT 2007 Goodbye World
- Mon Mar 26 15:28:47 EDT 2007 Hello World
- Mon Mar 26 15:28:47 EDT 2007 Goodbye World
- Mon Mar 26 15:28:48 EDT 2007 Goodbye World
- Mon Mar 26 15:28:48 EDT 2007 Hello World
- Mon Mar 26 15:28:49 EDT 2007 Goodbye World
- Mon Mar 26 15:28:49 EDT 2007 Hello World
- Mon Mar 26 15:28:50 EDT 2007 Goodbye World
- Mon Mar 26 15:28:50 EDT 2007 Hello World
- Mon Mar 26 15:28:51 EDT 2007 Goodbye World
- Mon Mar 26 15:28:51 EDT 2007 Hello World
- Mon Mar 26 15:28:52 EDT 2007 Hello World
- Mon Mar 26 15:28:52 EDT 2007 Goodbye World
- Mon Mar 26 15:28:53 EDT 2007 Hello World
```

- Two message sets (Hello and Goodbye) are not exactly interleaved.

29

## Exercise

- Modify HelloWorldTest2.java;

- Replace the following 4 lines

- Thread thread1 = **new** Thread(runnable1);
    - Thread thread2 = **new** Thread(runnable2);
    - thread1.start();
    - thread2.start();

- with the following 2 lines:

- runnable1.run();
    - runnable2.run();

- What does the program output?

31

- JVM's thread scheduler gives you NO guarantee about the order of thread execution.
- There are always slight variations in the time to execute a threaded task
  - especially when calling OS system calls (typically I/O related system calls)
- Expect that the order of thread execution is somewhat random.

30

- Modify HelloWorldTest2.java;

- Replace the following 4 lines

- Thread thread1 = **new** Thread(runnable1);
    - Thread thread2 = **new** Thread(runnable2);
    - thread1.start();
    - thread2.start();

- with the following 2 lines:

- runnable1.run();
    - runnable2.run();

- Output: “Hello World” in 10 lines, followed by “Good bye” in 10 lines.
  - The program runs **sequentially**, not concurrently.

32