OPERATING SYSTEMS

Threads and Concurrency 03

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 2

UNIT 2: Threads and Concurrency

Introduction to Threads, types of threads, Multicore Programming, Multithreading Models, Thread creation, Thread Scheduling, PThreads and Windows Threads, Mutual Exclusion and Synchronization: software approaches, principles of concurrency, hardware support, Mutex Locks, Semaphores. Classic problems of Synchronization: Bounded-Buffer Problem, Readers -Writers problem, Dining Philosophers Problem concepts. Synchronization Examples - Synchronisation mechanisms provided by Linux/Windows/Pthreads. Deadlocks: principles of deadlock, tools for detection and Prevention.

OPERATING SYSTEMS

Course Outline - Unit 2

13	4.1,4.2	Introduction to Threads, types of threads, Multicore Programming.	4	42.8
14	4.3,5.4	Multithreading Models, Thread creation, Thread Scheduling	4	
15	4.4	Pthreads and Windows Threads	4	
16	6.1-6.3	Mutual Exclusion and Synchronization: software approaches	6	
17	6.3-6.4	principles of concurrency, hardware support	6	
18	6.5,6.6	Mutex Locks, Semaphores	6	
19	6.7.1-6.7.3	Classic problems of Synchronization: Bounded-Buffer Problem, Readers -Writers problem, Dining Philosophers Problem concepts	6	
20	6.9	Synchronization Examples - Synchronisation mechanisms provided by Linux/Windows/Pthreads.	6	
21	Handouts	Demonstration of programming examples on process synchronization		
22	7.1-7.3	Deadlocks: principles of deadlock, Deadlock Characterization.	7	
23	7.4	Deadlock Prevention, Deadlock example	7	
24	7.6	Deadlock Detection	7	

Topics Outline

- Threads and Concurrency
  - Thread Libraries
  - Thread Creation
  - Thread Scheduling

Thread Libraries

- A **thread library** provides the programmer with an API for creating and managing threads.

- There are **two** primary ways of implementing a thread library.

  - The **first** approach is to provide a library entirely in **user** space with **no kernel** support.

    - **All** code and data structures for the library **exist** in user space.

    - This means that **invoking** a **function** in the library results in a **local** function call in **user space** and **not** a **system** call.

Thread Libraries

- A **thread library** provides the programmer with an API for creating and managing threads.

- There are **two** primary ways of implementing a thread library.

    - The **second** approach is to **implement** a **kernel-level** library supported **directly** by the **operating system**.

        - In this case, **code** and **data structures** for the library **exist** in **kernel space**.

        - Invoking a **function** in the API for the library typically **results** in a **system call** to the **kernel**.

Thread Libraries

- **Three** main thread libraries are in use today
  - **POSIX Pthreads**
  - **Window Threads**
  - **Java Threads**

Thread Libraries

- **Three** main thread libraries are in use today
  - **POSIX Pthreads**
    - **Pthreads**, the threads **extension** of the POSIX standard, may be provided as either a **user-level or** a **kernel-level** library.

Thread Libraries

- **Three** main thread libraries are in use today
  - **Windows Threads**
    - The **Windows** thread library is a **kernel-level** library available on Windows systems.

Thread Libraries

- **Three** main thread libraries are in use today
  - **Java Threads**
    - The **Java** thread API allows **threads** to be **created** and **managed directly** in **Java** programs.
    - However, because in most instances the **JVM** is running on **top** of a **host** operating system, the **Java thread API** is generally **implemented** using a **thread library** available on the **host** system.
    - This means that on **Windows systems**, **Java threads** are typically **implemented** using the **Windows API** ; **UNIX and Linux systems often use Pthreads**.

Thread Libraries

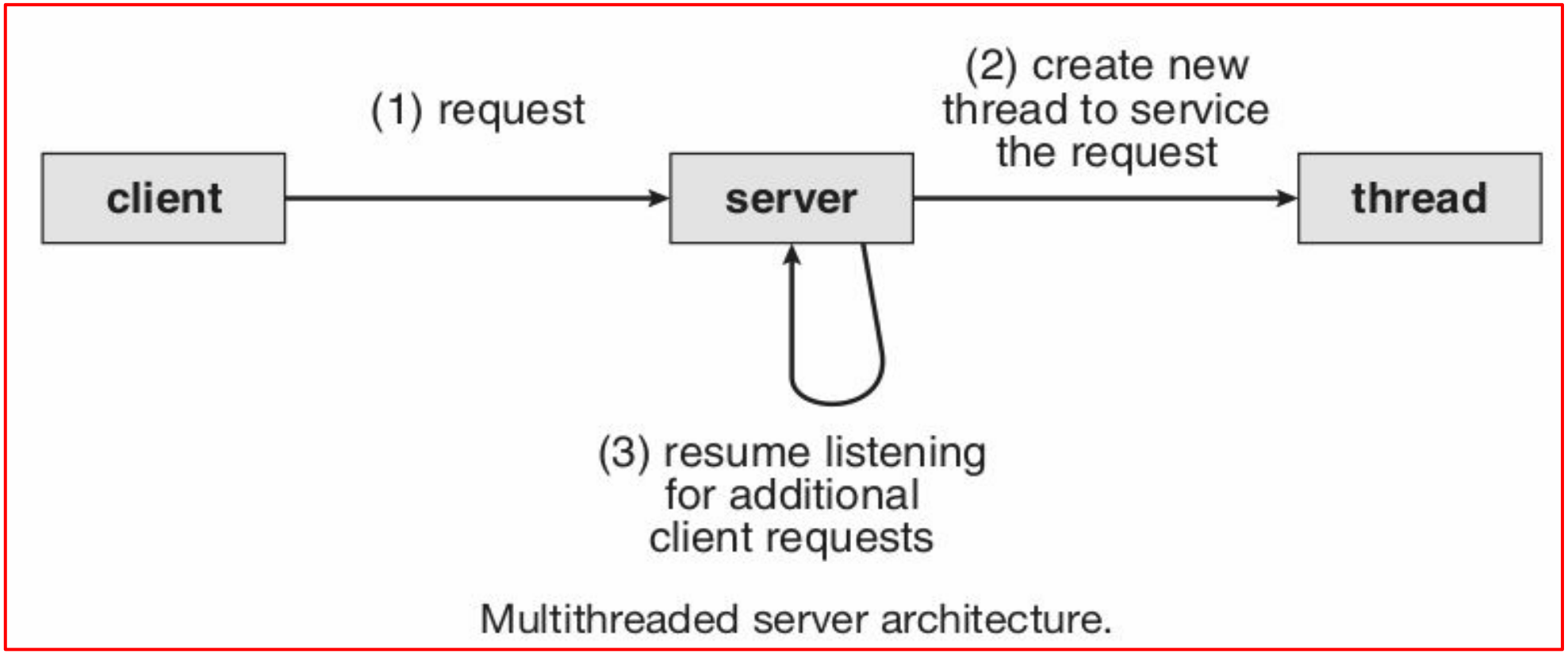
- For **POSIX** and **Windows threading**, any **data** declared globally that is, **declared outside** of any function are **shared** among all **threads** belonging to the **same** process.
- Since **Java** has **no notion** of global data, access to shared data must be **explicitly** arranged between threads.
- **Data** declared **local** to a function are typically stored on the **stack**.
- Since each thread has its own stack, each thread has its own copy of local data.

Thread Creation

- **Two** general strategies for creating multiple threads
  - **Asynchronous** Threading
    - Once the **parent creates** a **child thread**, the parent resumes its execution, so that the **parent** and **child** execute **concurrently**.
    - Each thread runs independently of every other thread, and the **parent** thread **need not** know when its **child** terminates.
    - Since the threads are **independent**, there is typically little **data** sharing between threads.

Thread Creation

- **Two** general strategies for creating multiple threads
  - **Asynchronous** Threading



Multithreaded server architecture.

Thread Creation

- **Two** general strategies for creating multiple threads
  - **Synchronous** Threading.
    - Synchronous threading **occurs** when the **parent** thread **creates one or more children** and then must **wait** for **all** of its **children** to **terminate** before it **resumes** using the **join** strategy.
    - Typically, synchronous threading involves significant **data sharing among threads**.

Thread Creation - Pthreads

- **Pthreads** refers to the **POSIX** standard (IEEE 1003.1c) defining an **API** for **thread creation** and **synchronization**.
  - This is a specification for thread behavior, not an implementation.
  - Operating-system **designers** may implement the specification in any way they wish.
  - **Numerous** systems **implement** the Pthreads specification; **most** are **UNIX** -type systems, including Linux, Mac OS X , and Solaris.
  - Although **Windows doesn't** support **Pthreads natively**, some **third party** implementations for Windows are **available**.

Thread Creation - Windows Threads

- The technique for creating threads using the Windows thread library is similar to the Pthreads technique in several ways.

- One must include the **windows.h** header file when using the Windows API

Thread Creation - Java Threads

- **Threads** are the **fundamental** model of **program** execution in a Java program, and the **Java** language and its **API** provide a rich set of features for the **creation** and **management** of **threads**.

- **All** Java programs comprise **at least** a **single** thread of control, even a simple Java program consisting of only a main() method runs as a single thread in the JVM .

- Java threads are **available** on any system that **provides** a **JVM** including Windows, Linux, and Mac OS X .

- The **Java thread** API is available for **Android** applications as well.

Thread Creation - Java Threads

- There are **two** techniques for creating threads in a Java program.
    - **One** approach is to create a new class that is derived from the Thread class and to override its run() method.
    - An **alternative** and more commonly used approach is to define a class that implements the Runnable interface.
- When a class implements Runnable , it must define a run() method.
- Creating a Thread object does not specifically create the new thread; rather, the start() method creates the new thread.
- Calling the start() method for the new object does two things:

1. It allocates memory and initializes a new thread in the JVM.
2. It calls the run() method, making the thread eligible to be run by the JVM .
    - Note again that we never call the run() method directly.
    - Rather, we call the start() method, and it calls the run() method on our behalf.

Thread Scheduling

- On operating systems that support threads, it is kernel-level threads not processes that are being scheduled by the operating system.

- User-level threads are managed by a thread library, and the kernel is unaware of them.

- To run on a CPU , user-level threads must ultimately be mapped to an associated kernel-level thread, although this mapping may be indirect and may use a lightweight process (LWP).

Thread Scheduling: Contention Scope

- One distinction between user-level and kernel-level threads lies in how they are scheduled.
- On systems implementing the many-to-one and many-to-many models, the thread library schedules user-level threads to run on an available LWP
- This scheme is known as **process contention scope (PCS)**, since **competition** for the **CPU** takes place among **threads belonging** to the **same process**.
- When we say the thread library schedules user threads onto available LWPs, we do not mean that the threads are actually running on a CPU.
- That would require the operating system to schedule the kernel thread onto a physical CPU.
- To **decide** which kernel-level thread to schedule onto a CPU , the kernel uses **system-contention scope** (SCS).

Thread Scheduling: Contention Scope

- Competition for the CPU with **SCS** scheduling takes place among **all threads** in the system.
- Systems using the **one-to-one** model, such as **Windows, Linux, and Solaris**, schedule threads using **only SCS**.
- Typically, **PCS** is done according to priority the scheduler selects the runnable thread with the highest priority to run.
- User-level thread priorities are set by the programmer and are not adjusted by the thread library, although some thread libraries may allow the programmer to change the priority of a thread.
- It is important to note that PCS will typically preempt the thread currently running in favor of a higher-priority thread; however, there is no guarantee of time slicing among threads of equal priority.

Thread Scheduling: Pthread Scheduling

- The POSIX Pthread API that allows specifying **PCS** or **SCS** during thread creation.
- Pthreads identifies the following contention scope values:
  - PTHREAD_SCOPE_PROCESS schedules threads using **PCS** scheduling.
  - PTHREAD_SCOPE_SYSTEM schedules threads using **SCS** scheduling.
- On systems implementing the many-to-many model, the **PTHREAD_SCOPE_PROCESS** policy schedules user-level threads onto available **LWPs**.
- The **PTHREAD_SCOPE_SYSTEM** scheduling policy will create and bind an LWP for each user-level thread on many-to-many systems, effectively mapping threads using the one-to-one policy

Topics Uncovered in this Session

- Threads and Concurrency
  - Thread Libraries
  - Thread Creation

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com