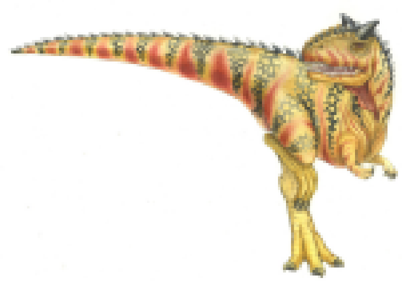
Chapter 4: Threads& Concurrency



Outline

- Overview

Objectives

- Multicore Programming
- Multithreading Models
- Thread Libraries
- Implicit Threading
- Threading Issues
- Operating System Examples

- Identify the basic components of a thread, and contrast threadsand processes
- Describe the benefits and challenges of designng multithreaded applications
- Illustrate different approaches to implicit threading includingthread pools, fork-join, and Grand Central Dispatch
- Describe how the Windows and Linux operating systems represent threads
- Design multithreaded applications using the Pthreads, Java, and Windows threading APIs

Threads

- A thread is a basic unit of CPU utilization. It consistsofathread ID, PC, a stack, and a set of registers.
- Traditional processes have a single threadof control.Itis also called as **heavyweight** process. Thereisoneprogram counter, and one sequence of instructionsthatcan be carried out at any given time.
- A multi-threaded application have multiplethreadswithina single process, each having their ownprogramcounter, stack and set of registers, but sharingcommoncode, data, and certain structures suchasopenfiles.Such processes are also called as **lightweight**processes.
- Process with multiple threads of control candomorethan one task at a time.

Motivation

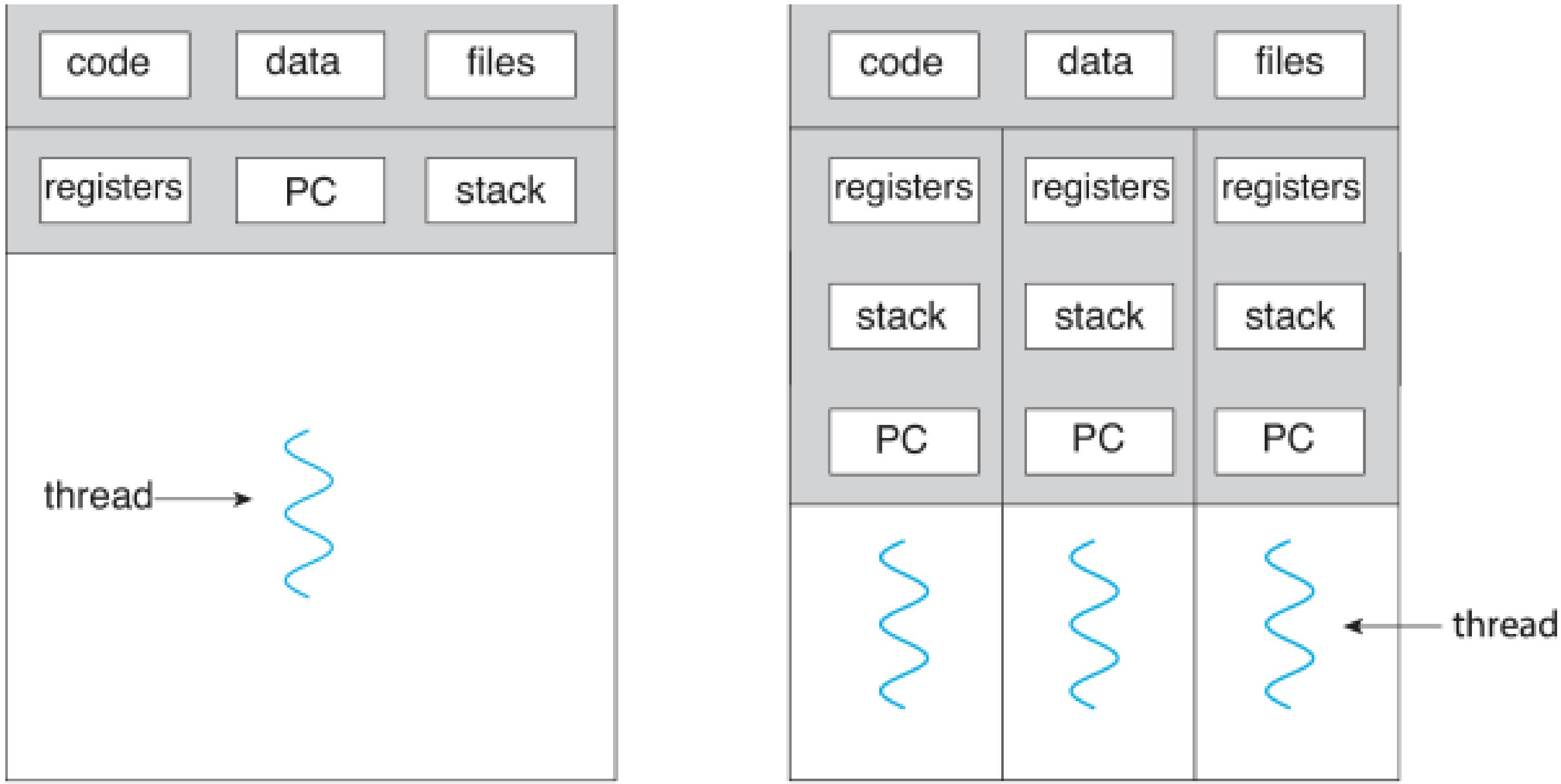
- Most modern applications are multithreaded
- If one of the tasks may

block, the other tasks can proceedwithout blocking.

- Threads run within application
- Multiple tasks with the application can be implementedbyseparate threads Eg. Word processor
  - Spell checking (background process)
  - process user input (foreground)
  - displaying graphics (another thread)
- Process creation is heavy-weight while thread creationislight-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

Single and MultithreadedProcesses

- In a heavyweight process, new processes are created to performtheworkin parallel (multiple processes running). Each heavyweight processcontains its own address space. Communication between these processeswould involve additional communications mechanisms such as IPC, RPC,sockets or pipes.



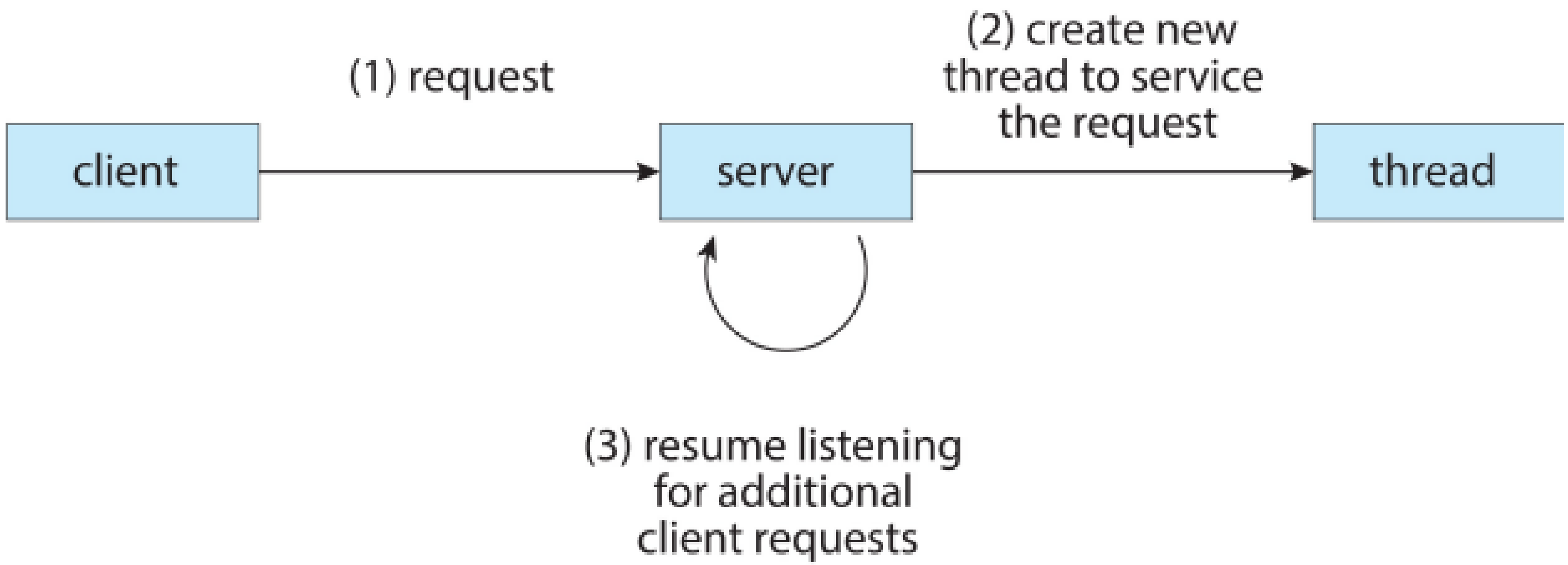
single-threaded process

multithreaded process

Multithreaded Server Architecture

Eg: Web

server



Benefits

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces Multi threading allows a program to continue running evenif part of it is blocked or is performing a lengthy operation, therebyincreasing responsiveness to the user.

- **Resource Sharing –** threads share resources of process, easierthan shared memory or message passing allows multiple tasks to be performed simultaneously inasingleaddress space.

- **Economy –** cheaper than process creation, thread switchinglower overhead than context switching Creating and managing threads is much faster Context switching between threads takes less time. ▪ **Scalability / utilization of multiprocessor architecture–**withmultitreading a process can take advantage of multicorearchitectures

Multicore Programming

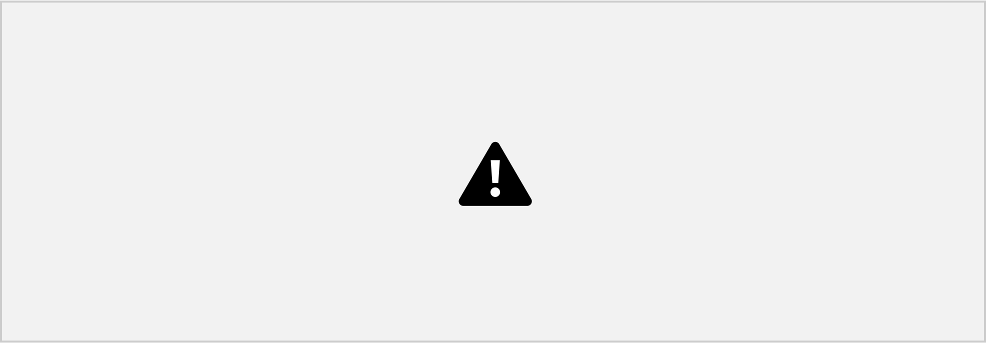
- A recent trend in computer architectureistoproducechips with multiple cores, or CPUs onasinglechip.▪ A multi-threaded application runningonatraditional**single-core** chip, would have to executethethreadsone after another.

- On a **multi-core chip**, the threads couldbespreadacross the available cores, allowingtrueparallelprocessing.
- For operating systems, multi-core chipsrequirenewscheduling algorithms to make better useofthemultiple cores available.

Concurrency vs. Parallelism

- **Concurrent execution on single-core system:**

- **Parallelism on a multi-core system:**



Multicore Programming

- **Multicore** or **multiprocessor** systems puttingpressureonprogrammers, challenges include:
  - **Dividing activities:** Examining applications to findactivitiesthatcan be performed concurrently.
  - **Balance:** All threads should have same level of computation; donot create thread for trivial tasks

- **Data splitting:** To prevent the threads frominterferingwithoneanother.
- **Data dependency:** If one task is dependent upon theresultsofanother, then the tasks need to be synchronized toassureaccess in the proper order.
- **Testing and debugging:** Inherently more difficult inparallel processing situations, as the race conditions becomemuchmorecomplex and difficult to identify.

Data and Task Parallelism

Types of parallelism

Data parallelism – distributes subsets of thesamedata across multiple cores, same operation oneach**Task parallelism** – distributing threads across cores, each thread performing unique operation

User Threads andKernel Threads

- Two types of threads are managed in modern systems: **User andKernel Threads**
- **User threads** - management done by user-level threads library. Theyare

supported above the kernel.

- creation, scheduling and management is done by library without kernelsupport.
- Problem: If kernel is single threaded, any user level threadperforminga blocking system call will block the entire process. ▪ Three primary thread libraries:
  - POSIX **Pthreads,** Windows threads, Java threads ▪ **Kernel threads** - Supported directly by the OS/Kernel allowingthekernel to perform multiple tasks simultaneously.
    - Generally slower to create and manage than user threads.
- Examples – virtually all general -purpose operating systems, including:•

Windows ,Linux, Mac OS X, iOS, Android 

User and Kernel Threads

MultithreadingModels

In a specific implementation, the user threads mustbemapped to kernel threads, using one of thefollowingmodels.

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

Many-to-One

- Thread management is handled by the thread library inuser space,which is very efficient.

- 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

▪ Overcomes the problems listed above involving blockingsystemcallsand the splitting of processes across multiple CPUs.

▪ Number of threads per process sometimes restricted duetooverheadof creating kernel threads. Hence places a limit on thenumber of threads created.

▪ Examples: Windows 95 to XP, Linux

Many-to-ManyModel

▪ Multiplexes any number of user threads onto an equal or smallernumber of kernel threads, combining the best features of theone-to-one and many-to-one models.

- Users have no restrictions on the number of threads created.
- Blocking kernel system calls do not block the entire process.
- Processes can be split across multiple processors.
- Individual processes may be allocated variable numbers of kernel threads, depending on the number of CPUs present andother factors.
- This model is also called as two-tier model.
- Eg: IRIX, HP-UX, and Tru64 UNIX.

Two-level Model

- Similar to M:M, except that it allows a user thread to be**bound**tokernel thread

ThreadingIssues

- Semantics of **fork()** and **exec()** system calls
- Thread cancellation of target thread

- Asynchronous or deferred
- Signal handling
  - Synchronous and asynchronous
- Thread pools
- Thread specific data / Thread-local storage
- Scheduler Activations

Semantics of fork() andexec()

- Does **`fork()`** duplicate only the calling thread or all threads?
  - Some UNIXes have two versions of fork
  - The new process can be a copy

of the parent, withall thethreads

- The new process is a copy of the single threadonly(thatinvoked the process)

- **`exec()`** usually works as normal – replacetherunningprocess including all threads i.e., if the threadinvokestheexec( ) system call, the program specified in theparametertoexec( ) will be executed by the thread created.

Thread Cancellation

- Terminating a thread before it has finished
- Thread to be canceled is **target thread**
- Example : Multiple threads required in loading a webpageissuddenlycancelled, if the browser window is closed.

- Two general approaches to cacel Threads that are no longer needed:•
  Asynchronous cancellation terminates the target threadimmediately
  - **Deferred cancellation** allows the target thread to periodicallycheck if it should be cancelled thus giving an opportunitytothethread, to terminate itself in an orderly fashion.
  - In this method, the operating system will reclaimall theresourcesbefore cancellation.

Thread Cancellation(Cont.)

- Invoking thread cancellation requests cancellation, but actual cancellation depends on thread state

- If thread has cancellation disabled, cancellation remains pendinguntilthread enables it
- Default type is deferred
- Cancellation only occurs when thread reaches **cancellationpoint**4 i.e.,

`pthread_testcancel()`

4 Then **cleanup handler** is invoked

- On Linux systems, thread cancellation is handled throughsignals

Signal Handling

- A signal is used to notify a process that a particular event hasoccurred.
- All signals follow same path-

1. A signal is generated by the occurrence of a particular event. 2. A generated signal is delivered to a process. 3. Once delivered, the signal must be handled. ▪ A signal can be invoked in 2 ways : synchronous or asynchronous. **Synchronous** signal – signal delivered to the same program. Eg–

illegal memory access, divide by zero error. **Asynchronous** signal – signal is sent to another program. Eg–Ctrl C▪ A signal can be handled by one of the two ways – **Default** signal handler - signal is handled by OS. **User-defined** signal handler - User overwrites the OShandler. It can

override default

Signal Handling(Cont.)

- In a single-threaded program, the signal is sent to the samethread.
- But, in multithreaded environment, the signal is deliveredinvarietyofways, depending on the type of signal – Where shouldasignal bedelivered for multi-threaded?
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for theprocess

ThreadPools

- In multithreading process, thread is created for every service. Eg–Inweb server, thread is created to service every client request. ▪ Creating new threads every time, when thread is neededandthendeleting it when it is done can be inefficient, as – ▪ Time is consumed in creation of the thread.
- A limit has to be placed on the number of active threads inthesystem.Unlimited thread creation may exhaust systemresources. ▪ Solution: Create a number of threads at process startupandplacethem in a pool where they await work
  - Threads are allocated from the pool when a request comes, andreturned to the pool when no longer needed(after the completionofrequest).
  - When no threads are available in the pool, the process mayhavetowait until one becomes available.

ThreadPools

- Advantages:
  - Usually slightly faster to service a request with an existingthreadthan create a new thread
  - Allows the number of threads in the application(s) tobeboundtothe size of the pool
  - Separating task to be performed from mechanics of creatingtaskallows different strategies for running task
    - 4 i.e.,Tasks could be scheduled to run periodically ▪ Windows API

supports thread pools:

Thread-Local Storage

- **Thread-local storage** (**TLS**) allows each thread to haveitsowncopyof data
- Useful when you do not have control over the thread creationprocess(i.e., when using a thread pool)
- Data of a thread, which is not shared with other threads iscalledthreadspecific data.
- Most major thread libraries (pThreads, Win32, Java) providesupportfor thread-specific data.
- Example – if threads are used for transactions and eachtransactionhas an ID. This unique ID is a specific data of the thread.
- Similar to `static` data
  - TLS is unique to each thread

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Typically use an intermediate data structure between user and kernel threads – **lightweight process** (**LWP**)
  - Appears to be a virtual processor on which process can schedule user thread to run
  - Each LWP attached to kernel thread

- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the **upcall handler** in the thread library
- This communication allows an application to maintain the correct number kernel threads

Thread Libraries

- **Thread library** provides programmer with API for creatingandmanaging threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creationandsynchronization
- ***Specification***, not ***implementation*** ▪ API specifies behavior of the

thread library, implementationisuptodevelopment of the library

- Common in UNIX operating systems (Linux & Mac OSX)

Pthreads Example

Pthreads Example(Cont.)

Pthreads Code for Joining10Threads

Endof Chapter4