

## CS 370: OPERATING SYSTEMS

### [THREADS]

#### At a glance: User vs Kernel threads

User level threads are hidden from the kernel's sight  
And need a runtime to function right  
Programmers face the lockout strain  
Restricted to the process' resources, a painful gain

Kernel threads are known to the kernel's core  
Schedulable, they open the door  
To access resources system wide  
More work for the kernel, with less time to bide

Shrideep Pallickara  
Computer Science  
Colorado State University

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

1

## Frequently asked questions from the previous class survey

- A process with multiple threads can run on multi-core and single-core systems?
- If a 2 processes A and B create shared memory, can threads from those processes access that memory?
- Can you think of threads as child processes?
- Do threads always run concurrently, or do they ever block until the nother thread finishes or returns something?
- Is the number of threads a program can create limited by the number of underlying cores?
- Say the main thread creates threads T1 and T2. When thread T1 is running, can T2 and the main thread also be running?
- When writing threads, do you specify the number of cores to use?
- Hyperthreading and its relation to execution of threads
- What if I call fork() in a process that has a 1000 threads, and don't invoke exec()?
- If threads are so great, why even use a process?



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.2

2

## Topics covered in this lecture

- User- and kernel-level threads
- Thread Models
- Thread Libraries



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.3

3

There's a thread you follow. It goes among  
things that change. But it doesn't change.  
People wonder about what you are pursuing.  
You have to explain about the thread.  
But it is hard for others to see.  
While you hold it you can't get lost.

*The Way It Is, William Stafford*



**WRITING MULTITHREADED  
PROGRAMS**

COMPUTER SCIENCE DEPARTMENT

4

## Going about writing multithreaded programs [1/2]

- The key idea is to write a **concurrent program** — one with many simultaneous activities
  - As a set of sequential streams of execution, or threads, that interact and share results in very precise ways
- **Subdivide** functionality into multiple separate & concurrent tasks
- Threads let us define a set of tasks that run concurrently *while* the code for each task is sequential



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.5

5

## Going about writing multithreaded programs [2/2]

- Managing **data** manipulated by tasks
  - Split to run on separate cores. BUT
    - Examine data dependencies between the tasks
- Threaded programs on multi-core systems have many different **execution paths**
  - Which may or may not reveal **bugs**
  - Testing and debugging is inherently harder



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

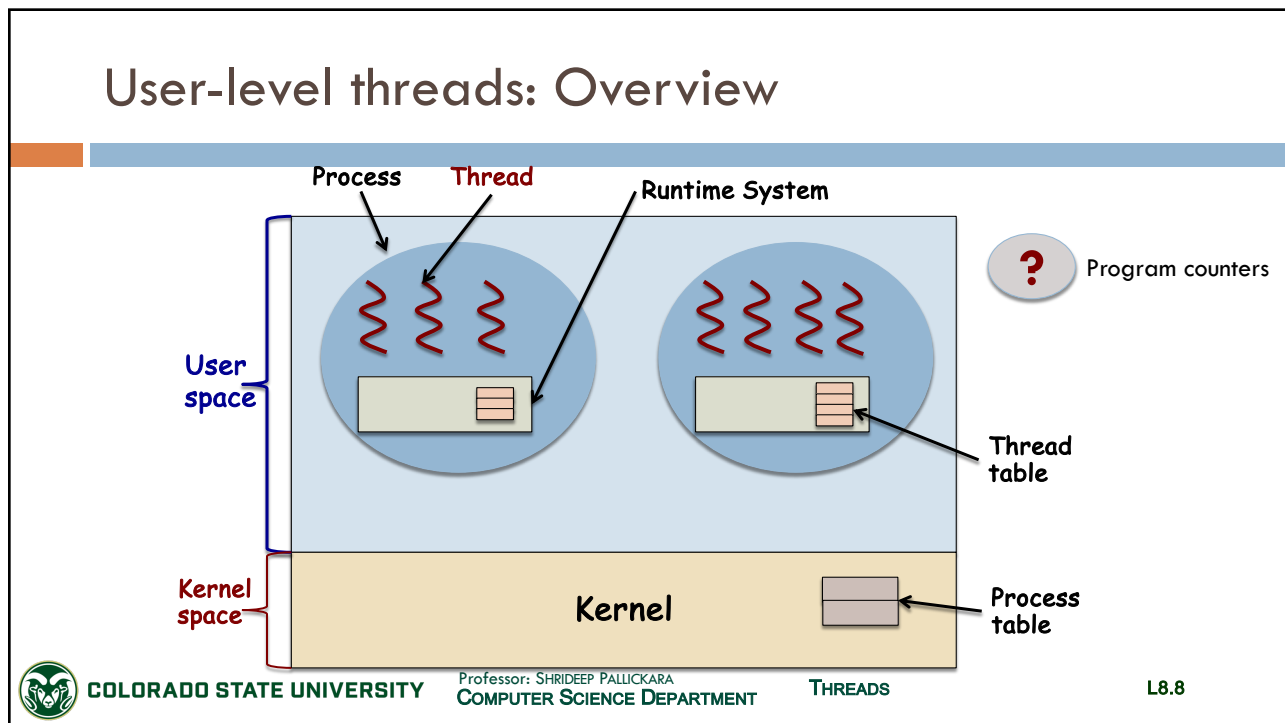
THREADS

L8.6

6



7



8

## User threads are invisible to the kernel and have low overhead

- **Compete among themselves** for resources allocated to their encapsulating process
- Scheduled by a *thread runtime* system that is **part** of the process code
- Programs link to a special library
  - Each library function is enclosed by a **jacket**
  - Jacket function calls thread runtime to do thread management
    - Before (and possibly after) calling jacketed library function



9

## User level thread libraries: Managing blocking calls

- **Replace** potentially blocking calls with non-blocking ones
- If a call does not block, the runtime invokes it
- If the call *may block*
  - ① Place thread on a list of *waiting* threads
  - ② Add call to list of actions to *try later*
  - ③ Pick another thread to run
- ALL control is **invisible** to user and OS



10

## Disadvantages of the user level threads model [1/2]

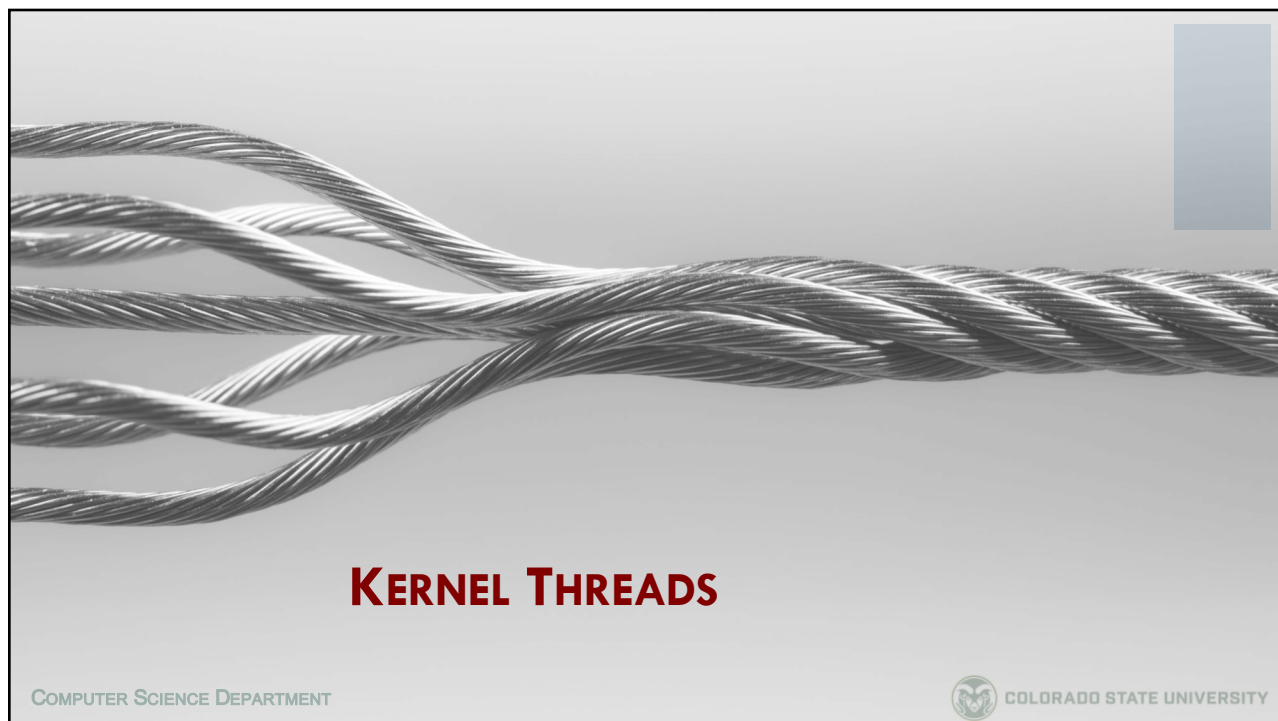
- Assumes that the runtime will **eventually regain** control, this is thwarted by:
  - CPU bound threads
  - Thread that *rarely* performs library calls ...
    - Runtime can't regain control to schedule other threads
- Programmer must avoid **lockout** situations
  - Force CPU-bound thread to *yield* control



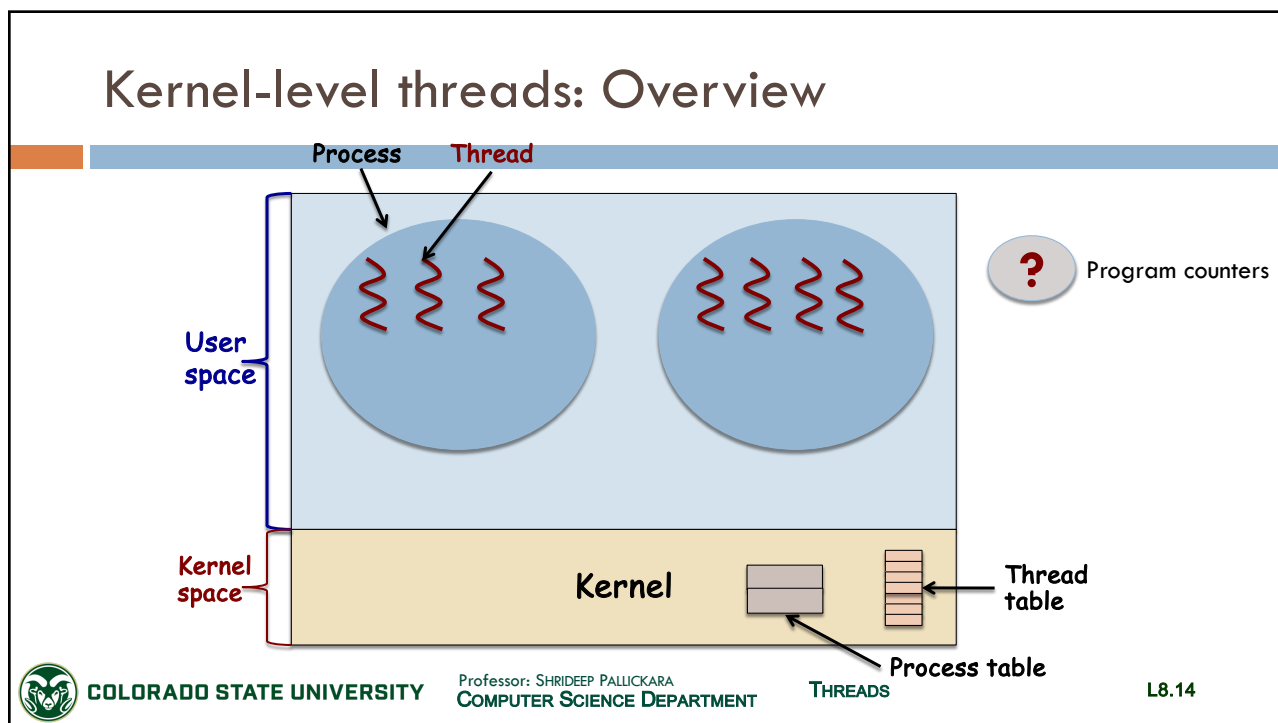
## Disadvantages of the user level threads model [2/2]

- Can only share processor resources allocated to encapsulating process
  - **Limits** available parallelism





13



14

## Kernel threads

- Kernel is aware of kernel-level threads as **schedulable entities**
  - Kernel maintains a thread table to keep track of all threads in the system
- **Compete system wide** for processor resources
  - Can take advantage of multiple processors



## Kernel threads: Management costs

- Scheduling is **almost as expensive** as processes
  - Synchronization and data sharing **less expensive** than processes
- More expensive to manage than user-level threads



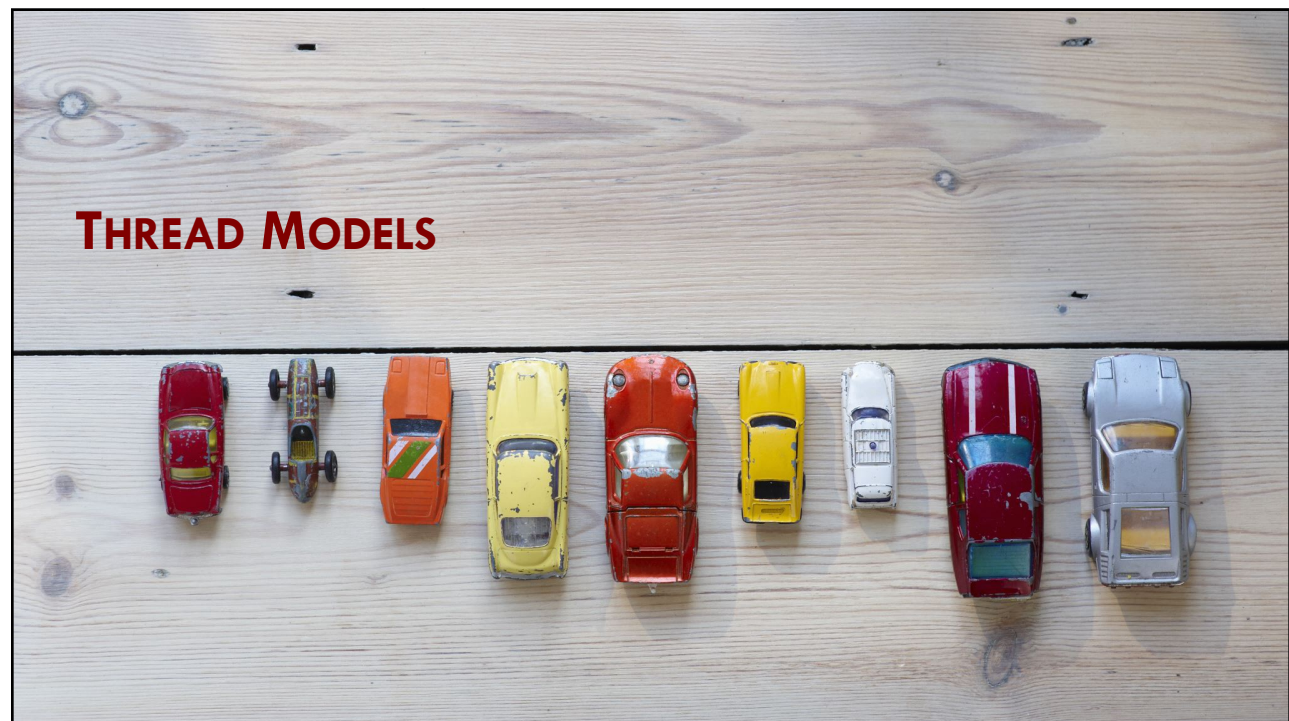


## Hybrid thread models

- Write programs in terms of user-level threads
- Specify number of schedulable entities associated with process
  - ▣ **Mapping at runtime** to achieve parallelism
- Level of user-control over mapping
  - ▣ Implementation dependent

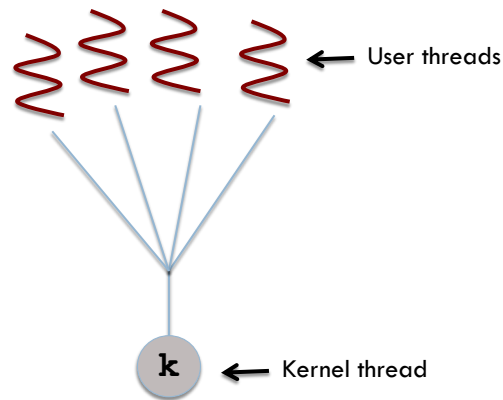


17



18

## The Many-to-One threading model



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.19

19

## Many-to-One Model maps many user-level threads to 1 kernel thread

- Thread management done by thread library in **user-space**
- What happens when one thread makes a *blocking system call*?
  - ▣ The entire process blocks!



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.20

20

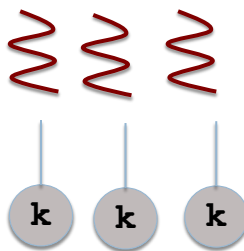
## Many-to-One Model maps many user-level threads to 1 kernel thread

- Only 1 thread can access kernel at a time
  - ▣ Multiple threads **unable** to run in parallel on multi-processor/core system
- E.g.: Solaris Green threads, GNU Portable threads



21

## The One-to-One threading model



22

## One-to-One Model:

### Maps each user thread to a kernel thread

- More **concurrency**
  - Another thread can continue to run, when a thread invokes a blocking system call
- Threads run in **parallel** on multiprocessors

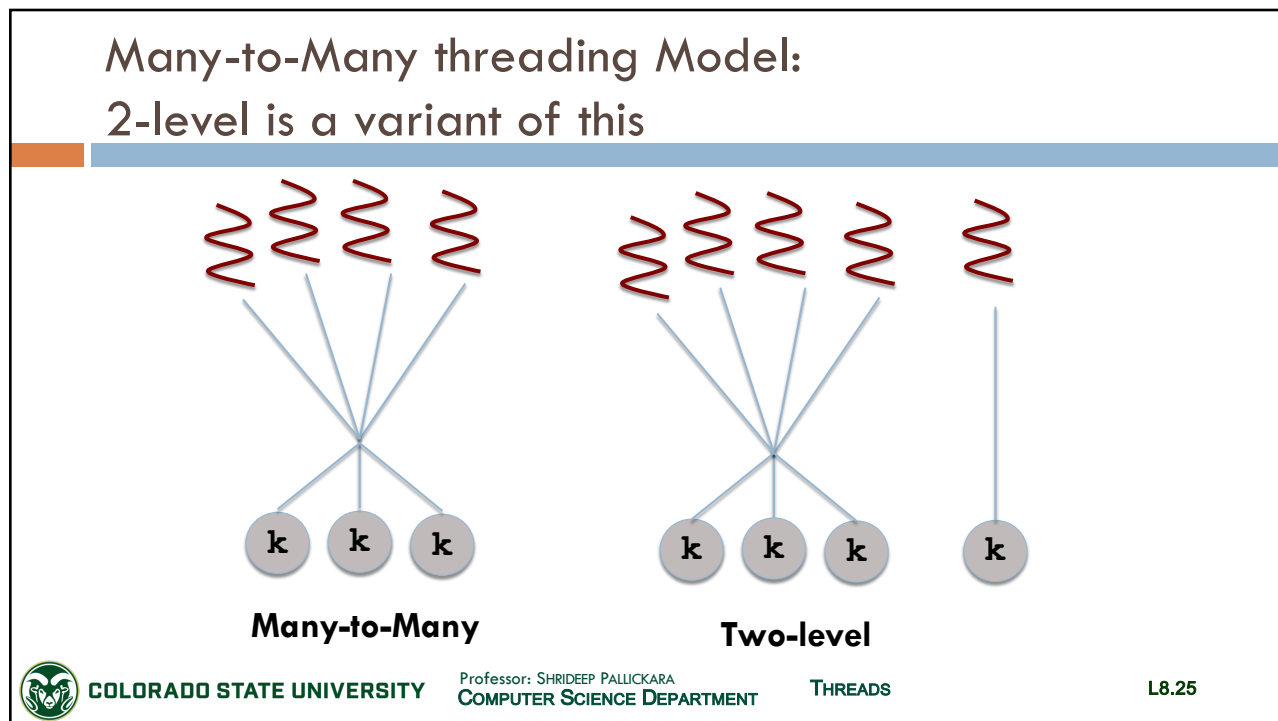


## One-to-One Model:


### Maps each user thread to a kernel thread

- Disadvantages:
  - There is an **overhead** for kernel thread creation
    - Multiple user threads can degrade application performance
- Supported by:
  - Linux
  - Windows family: NT/XP/2000/Vista/7/8/10/11
  - Solaris 9 and up





25

- ### Many-to-Many model
- ❑ **Multiplex** many user-level threads on a smaller number of kernel threads
  - ❑ Number of kernel threads may be specific to
    - ❑ Particular application
    - ❑ Particular machine
  - ❑ Supported in
    - ❑ IRIX, HP-US, and Solaris (prior to version 9)
-  **COLORADO STATE UNIVERSITY**    Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT    THREADS                      L8.26

26

## A comparison of the three models

	Many-to-one	One-to-One	Many-to-Many
<b>True Concurrency</b>	NO	YES	YES
<b>During blocking system call?</b>	Process Blocks	Process DOES NOT block	Process DOES NOT block
<b>Kernel thread creation</b>	Kernel thread already exists	Kernel thread creation overhead	Kernel threads available
<b>Caveat</b>	Use system calls (blocking) with care	Don't create too many threads	



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.27

27

Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry.

Richard P. Feynman

**THREAD LIBRARIES**  
*Provide an API for creating and managing threads*

28

## Thread libraries provide an API for managing threads

- Includes functions for :
  - ① Thread creation and destruction
  - ② Enforcement of mutual exclusion
  - ③ Conditional waiting
- Runtime system to manage threads
  - Users are **not aware** of this



## User level thread libraries

- **No kernel support**
- Library code & data structures reside in user space
- Invoking a library function **does not** result in a system call
  - Local function call in user space



## Kernel level thread libraries

- Library code & data structures in kernel space
- Invoking library function typically **results in a system call**



31

## Thread libraries provide an API for creating and managing threads

	User level library	Kernel level library
Library code and data structures	Reside in user space	Reside in kernel space
Can invocation of library function result in system call?	NO	YES
OS support	NO	YES



32



## Dominant thread libraries (1)

- POSIX pthreads
  - ▣ Extends POSIX standard (IEEE 1003.1c)
  - ▣ Provided as user- or kernel-level library
  - ▣ Linux, Mac OS X, Solaris, BSD
- Win32 thread library
  - ▣ Kernel-level library



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.33

33

## Dominant thread libraries (2)

- Java threading API
  - ▣ Implemented using **thread library on host system**
    - On Windows: Threads use Win32 API
    - UNIX/Linux: Uses pthreads



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.34

34




35

## Java threads example

- We will use a thread to perform summation of a non-negative integer

$$sum = \sum_{i=0}^N i$$

- If  $N=5$ , we compute the sum of 0 through 5
  - $0 + 1 + 2 + 3 + 4 + 5 = 15$

 **COLORADO STATE UNIVERSITY** Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.36

36

## Java

- Designed from the ground-up to support **concurrent** programming
  - Basic concurrency support in the language and class libraries
- Java 1.5 (or 5) and higher
  - Powerful high-level concurrency APIs



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.37

37

## JVMs harness the thread models of the host OS

- Windows/Linux have a one-to-one model
  - So a thread maps to a kernel thread
- Tru64 UNIX uses the many-to-many model
  - Java threads mapped accordingly
- Solaris
  - Initially, used Green Threads → many-to-one
  - Version 9 onwards: one-to-one model



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.38

38

## Creating Threads in Java

- ① Create a new class **derived** from Thread
  - ▣ Override its `run()` method
- ② More commonly used: `Runnable` interface
  - ▣ Has 1 method `run()`
  - ▣ Create new Thread class by passing a `Runnable` object to its constructor
- ③ The `Executor` interface (`java.util.concurrent`)
  - ▣ Has 1 method `execute()`



## Java Threads: Interrupts

- ▣ Invoke `interrupt()` on the Thread
- ▣ Threads must support their **own** interruption
- ▣ An **interruptible thread** needs to
  - ① Catch the `InterruptedException`
    - ▣ Methods such as `sleep()` throw this, and are designed to cancel the operation and return
  - ② Periodically invoke `Thread.interrupted()` to see if it has been interrupted



## Java Threads: joins

- If thread object `threadA` is currently executing
- Another thread can call `threadA.join()`
  - Causes current thread to pause execution **until threadA terminates**
- Variants of `join()`
  - Specify a waiting period



## Using Java Threads

[1/3]

```
class Sum {  
    private int sum;  
  
    public int get() {  
        return sum;  
    }  
  
    public void set(int sum) {  
        this.sum = sum;  
    }  
}
```



## Using Java Threads

[2/3]

```
class Summation implements Runnable {
    private int upper;
    private Sum sumValue;

    public Summation(int upper, Sum sumValue) {
        this.upper = upper;
        this.sumValue = sumValue;
    }

    public void run() {
        int sum = 0;
        for (int i = 0; i <= upper; i++)
            sum += i;

        sumValue.set(sum);
    }
}
```



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.43

43

## Using Java Threads

[3/3]

```
public class Driver {
    public static void main(String[] args) {

        Sum sumObject = new Sum();
        int upper = Integer.parseInt(args[0]);

        Thread worker = new Thread(new Summation(upper, sumObject));
        worker.start();
        try {
            worker.join();
        } catch (InterruptedException ie) {
            ie.printStackTrace()
        }
        System.out.println("The sum of " + upper + " is " +
            sumObject.get());
    }
}
```



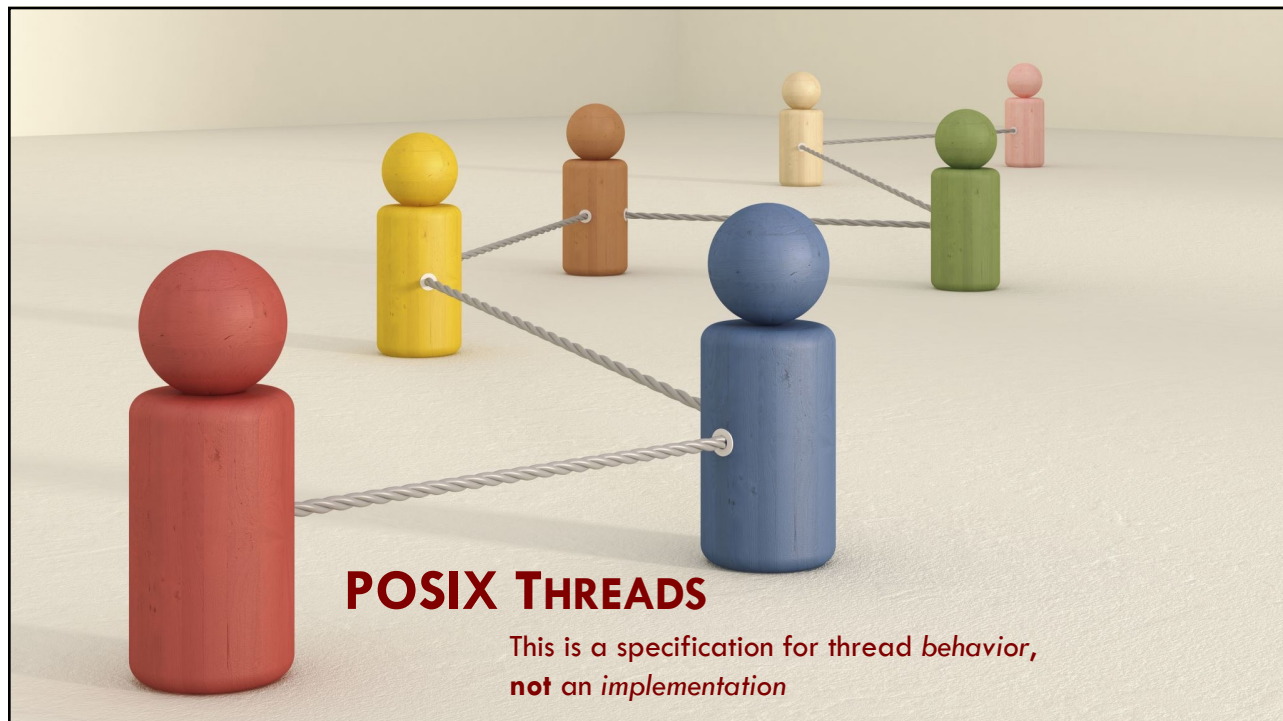
COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.44

44



45

## POSIX thread management functions: Return 0 if successful

POSIX function	Description
pthread_cancel	Terminate another thread
pthread_create	Create a thread
pthread_detach	Set thread to release resources
pthread_exit	Exit a thread without exiting process
pthread_kill	Send a signal to a thread
pthread_join	Wait for a thread
pthread_self	Find out own thread ID

Functions return a non-ZERO error code  
Do NOT set `errno`



46

## POSIX: Thread creation

`pthread_create()`

- Automatically makes the thread runnable *without* a start operation
- Takes 3 parameters:
  - ① Points to **ID** of newly created thread
  - ② **Attributes** for the thread
    - Stack size, scheduling information, etc.
  - ③ Name of **function** that the thread calls when it begins execution



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.47

47

## Using Pthreads (1)

```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */

void *runner(void *param); /* the thread */
```



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.48

48



## Using Pthreads (2)

```
int main(int argc, char *argv){  
  
    pthread_t tid;    pthread_attr_t attr;  
    /* get the default attributes */  
    pthread_attr_init(&attr);  
  
    /* create the thread */  
    pthread_create(&tid, &attr, runner, argv[1]);  
  
    /* now wait for the thread to exit */  
    pthread_join(tid, NULL);  
  
    printf("sum = %d\n", sum);  
}
```



## Using Pthreads (3)

```
/**  
 * The thread will begin control in this function  
 */  
void *runner(void *param)  
{  
    int i, upper = atoi(param);  
    sum = 0;  
  
    if (upper > 0) {  
        for (i = 1; i <= upper; i++)  
            sum += i;  
    }  
  
    pthread_exit(0);  
}
```



## POSIX: Detaching and Joining

- When a thread exits it does not release its resources
  - Unless it is a **detached thread**
- `pthread_detach()`
  - Sets *internal options* to specify that storage for thread can be **reclaimed** when it exits
  - 1 parameter: Thread ID of the thread to detach



51

## POSIX: Thread joins

- Threads that are not detached are *joinable*
- Undetached threads don't release resources until
  - Another thread calls `pthread_join` for them
  - Process exits
- `pthread_join`
  - Takes ID of the thread to **wait** for
  - **Suspends** calling thread till target terminates
  - Similar to `waitpid` at the process level
  - `pthread_join(pthread_self())`?
    - **Deadlock!**



52

## POSIX: Exiting and cancellation

- If a process calls `exit`, **all** threads terminate
- Call to `pthread_exit` causes only the calling thread to terminate
- Threads can force other threads to return through a **cancellation** mechanism
  - `pthread_cancel`: takes thread ID of target
  - Depends on **type** and **state** of thread



## More info on `pthread_cancel`

- **State:** `pthread_setcancelstate` to change state
  - `PTHREAD_CANCEL_ENABLE`
  - `PTHREAD_CANCEL_DISABLE`
    - Cancellation requests are held pending
- **Cancellation type** allows thread to control when to exit
  - `PTHREAD_CANCEL_ASYNCHRONOUS`
    - Any time
  - `PTHREAD_CANCEL_DEFFERED`
    - Only at specified cancellation points



## Win32 Threads

- CreateThread
  - ▣ Security Information, size of stack, flag (start in suspended state?)
- WaitForSingleObject
- CloseHandle



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.55

55

## The contents of this slide-set are based on the following references

- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9<sup>th</sup> edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330. [Chapter 4]*
- *Andrew S Tanenbaum and Herbert Bos. Modern Operating Systems. 4<sup>th</sup> Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 2].*
- *Kay Robbins & Steve Robbins. Unix Systems Programming, 2nd edition, Prentice Hall ISBN-13: 978-0-13-042411-2. [Chapter 12]*
- *Thomas Anderson and Michael Dahlin. Operating Systems: Principles and Practice, 2<sup>nd</sup> Edition. Recursive Books. ISBN: 0985673524/978-0985673529. [Chapter 4]*



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

THREADS

L8.56

56