中国科学技术大学
University of Science and Technology of China

011174.01: Operating System
操作系统原理与设计

Chapter 4: Threads & Concurrency

陈香兰(xlanchen@ustc.edu.cn)
高能效智能计算实验室，CS，USTC @ 合肥
嵌入式系统实验室，CS，USTC @ 苏州

温馨提示：

为了您和他人的工作学习，
请在课堂上**关机或静音**。

不要在课堂上**接打电话**。

Chapter Objectives

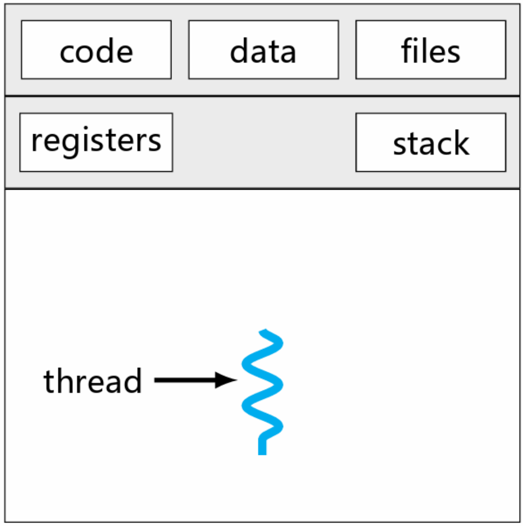
- Chapter Objectives
  1. **To introduce the notion of a thread** – a fundamental unit of CPU utilization that forms the basis of multithreaded computer system.
  2. **To discuss the APIs** for Pthreads, Win32, and JAVA **thread libraries**.
  3. Some threading issues.

Outline

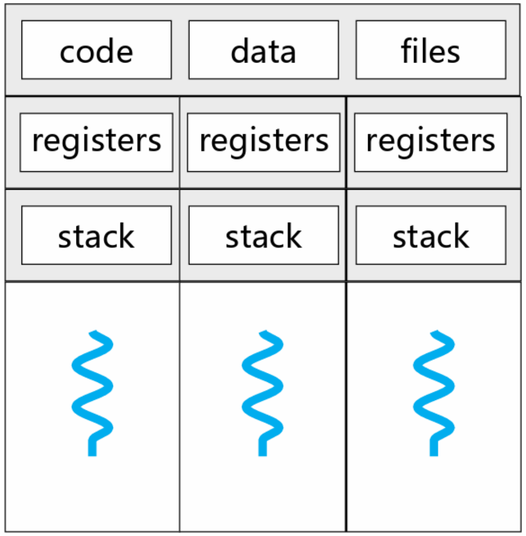
- Overview
  - Single-threaded and multithreaded processes
  - Motivation
  - Benefits
- Multicore programming
- Multithreading models
- Thread Libraries
- Threading Issues
- OS examples for thread
- Threading Scheduling

Thread concept overview

- A **thread** is a **basic unit of CPU utilization**;
  - It comprises a thread **ID**, a **program counter**, a **register set**, and a **stack**.
  - It shares with other threads belonging to the same process the code section, the data section, and other OS resources, such as open files, signals, etc.
- A **traditional process** has a single thread of control: **heavyweight process**.



Single-threaded process



Multithreaded process

Motivation

- On modern desktop PC, **many APPs are multithreaded**.
  - **A separate process with several threads**
  - Example 1: A web browser
    - One for displaying images or text;
    - another for retrieving data from network
  - Example 2: A word processor
    - One for displaying graphics;
    - Another for responding to keystrokes from the user;
    - And a third for performing spelling & grammar checking in the background
  - Example 3: RPC servers
    - For each message, a separate thread is used to service the message
    - concurrency↑

Motivation (cont.)

中国科学技术大学 University of Science and Technology of China

- Motivation
  - In certain situations, a single application may be required to perform several similar tasks.
    - Example: a web server
  - Allow a server to service several concurrent requests.
    - Example: an RPC server and Java's RMI systems
  - The OS itself needs to perform some specific tasks in kernel, such as managing devices or interrupt handling.
    - **PARTICULAR, many OS systems are now multithreaded**.
    - Example: Solaris, Linux

Benefits

1. Responsiveness (响应性)
   - Example: an interactive application such as web browser
     - while one thread loading an image, another allowing user interaction
2. Resource Sharing
   - address space, memory, and other resources
3. Economy (经济，指开销)
   - Example: Solaris
     - creating a process is about 30 times slower then creating a thread;
     - context switching is about 5 times slower
4. Scalability (可伸缩性)
   - Utilization of MP Architectures;
   - Parallelism and concurrency ↑

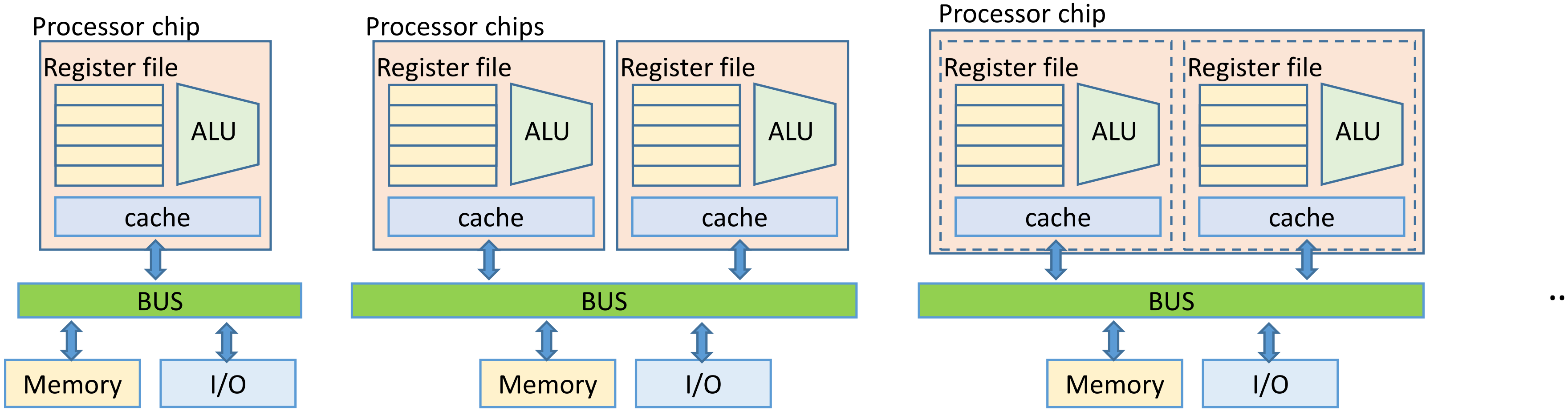
Outline

- Overview
- **Multicore programming**
    - Multicore and hyper-threading
    - Concurrency and parallelism
    - Programming challenges
    - Types of parallelism
- Multithreading models
- Thread Libraries
- Threading Issues
- OS examples for thread
- Threading Scheduling

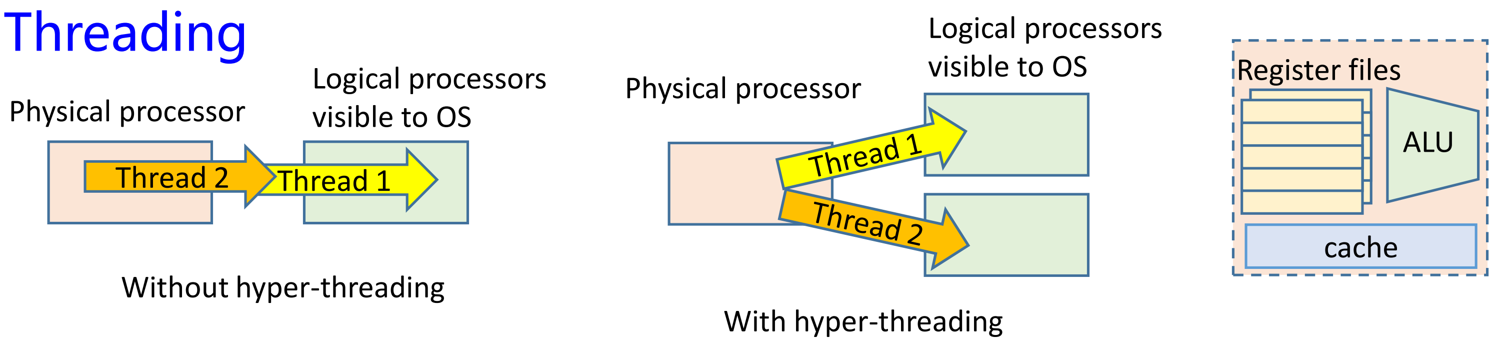
Multicore programming

中国科学技术大学
University of Science and Technology of China

- Single-CPU VS. Multi-CPU Single-core VS. Multicore



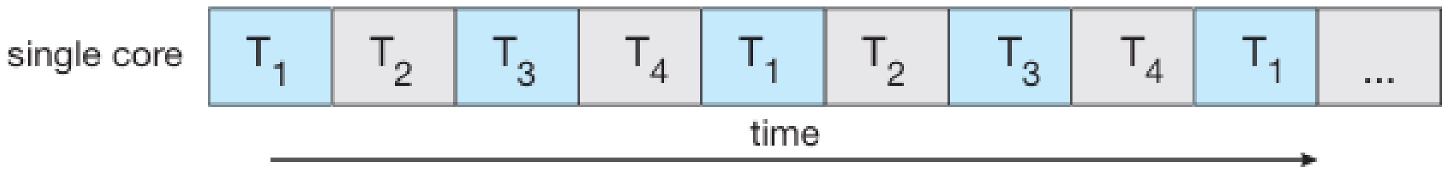
- Hyper-Threading



- **Multithreaded programming** provides a **mechanism** for **more efficient use** of these multiple computing resources and **improved concurrency**.

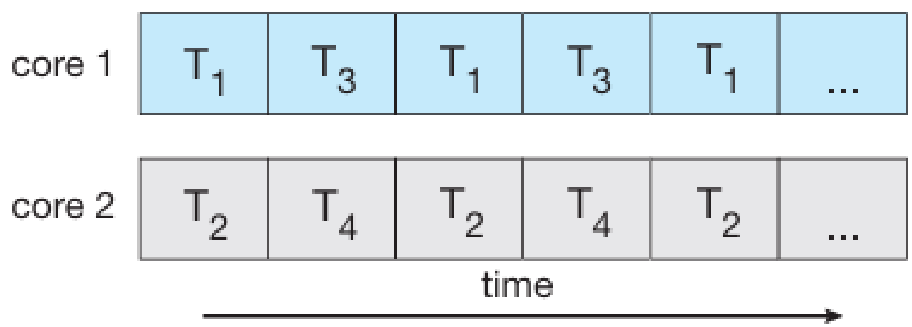
Multicore programming

- Concurrency VS parallelism



Interleaved execution on a single-core system

concurrency without parallelism



Parallel execution on a multicore system

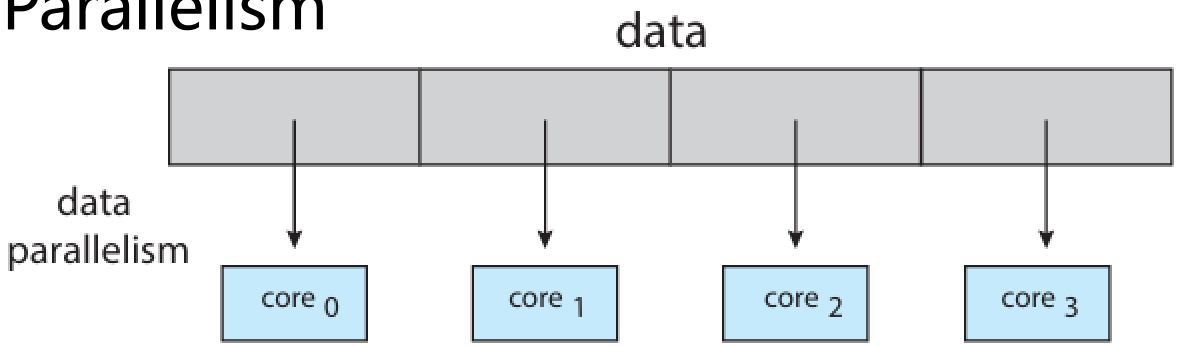
concurrency with parallelism

Multicore programming

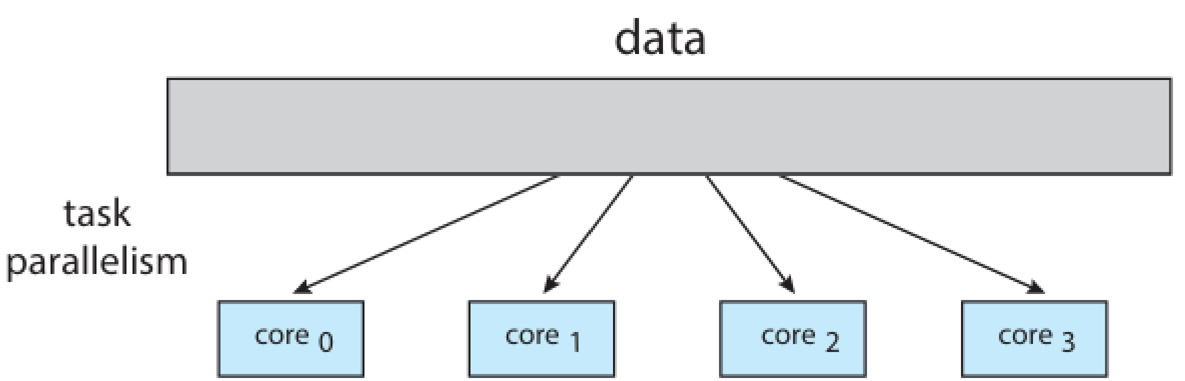
中国科学技术大学 University of Science and Technology of China

- Challenges in making better use of the multiple computing resources：
  - **OS designers**: What scheduling algorithms?
  - **APP programmers**: How to modify existing programs or to design new programs to be multithreaded?
    - **Identifying tasks**: how to divide APP into separate, concurrent tasks that can run in parallel?
    - **Balance**: how to ensure that the tasks perform equal work of equal value? Is each task worth the cost using a separate execution resource?
    - **Data splitting**: How to divide the data accessed and manipulated by the tasks to run on separate execution resources?
    - **Data dependency**: Any data dependencies among the tasks? How to examine? How to ensure the synchronized execution of the tasks?
    - **Testing & debugging**: How?

- Types of Parallelism



Data parallelism



Task parallelism

Data and task parallelism are not mutually exclusive, and an application may in fact use a **hybrid** of these two strategies.

Outline

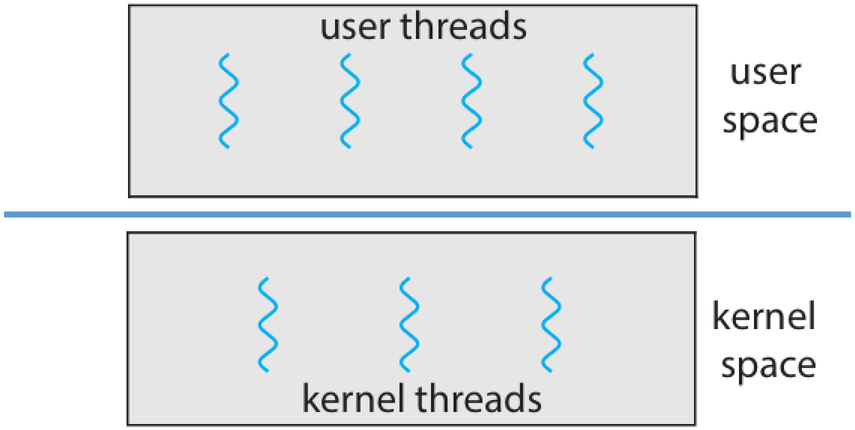
中国科学技术大学 University of Science and Technology of China

- Overview
- Multicore programming
- **Multithreading models**
  - User threads VS kernel threads
  - Multithreading models
- Thread Libraries
- Threading Issues
- OS examples for thread

Multithreading Models

中国科学技术大学 University of Science and Technology of China

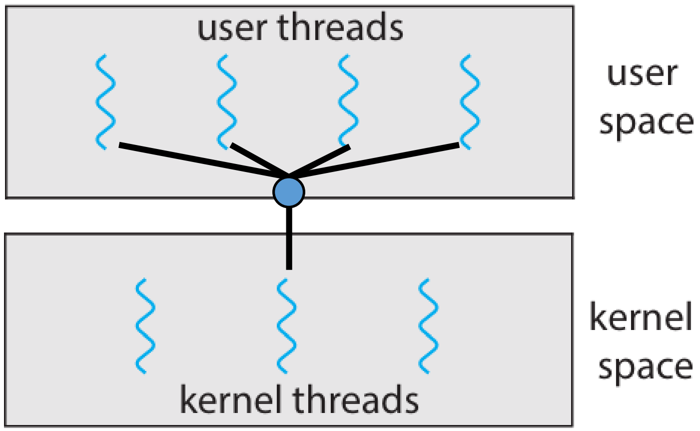
- Two methods to support threads
  - At the user level: User threads
    - Supported above the kernel; managed by user-level threads library without kernel support; Kernel may be multithreaded or not
    - Three primary thread libraries:
      1. POSIX Pthreads
      2. Win32 threads
      3. Java threads



  - By kernel: Kernel threads
    - Supported and managed directly by the OS.
    - Example: Virtually all contemporary OSes including
      - Windows, Linux, and macOS, …

Multithreading Models

- The relationship between user threads and kernel threads
  1. Many-to-One [n:1]
  2. One-to-One [1:1]
  3. Many-to-Many [n:m]



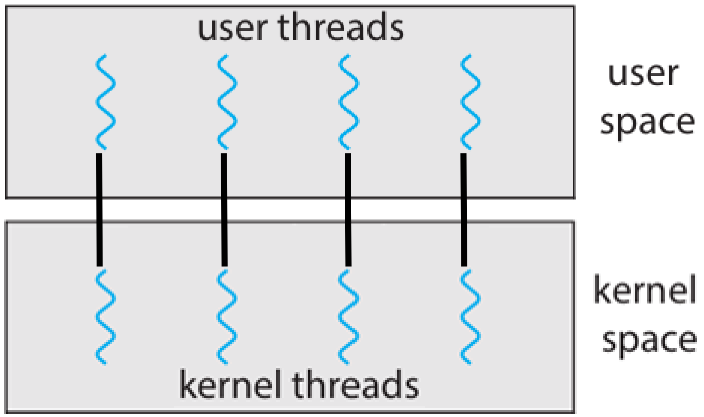
1. Many-to-One [n:1]

- Many user-level threads mapped to one kernel thread
- Examples:
  - Solaris Green Threads (adopted in early versions of Java)
  - GNU Portable Threads
- Disadvantages
  - the entire process will be blocked if a thread makes a blocking system call
  - unable to run in parallel on systems with multiple computing resources
- Very few used

Multithreading Models

2. One-to-One [1:1]

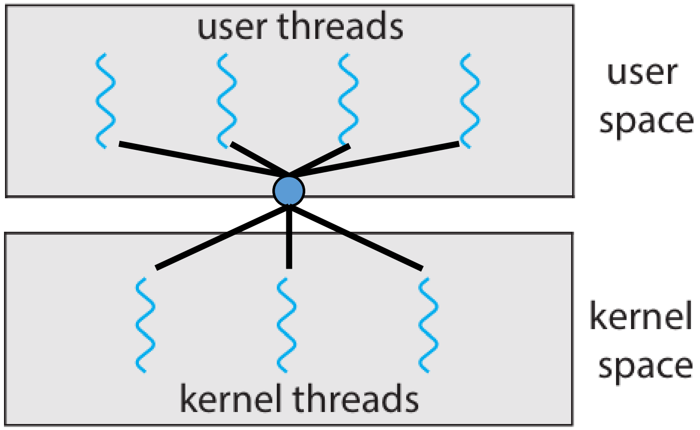
- Each user-level thread maps to one kernel thread
- More concurrency, but not economy, too many kernel threads may burden the performance of a system.
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and late



Multithreading Models

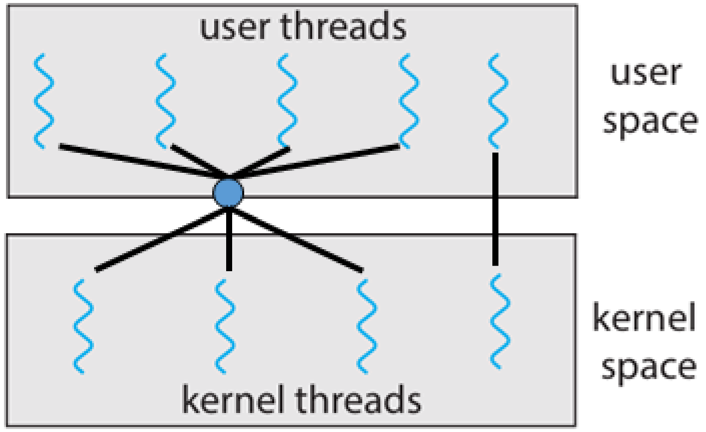
3. Many-to-Many [n:m]

- Allows many user level threads to be mapped to a smaller or equal number of kernel threads
- The number of kernel threads may be specific to either a particular APP or a particular machine.
- Examples
  - Solaris prior to version 9
  - Windows NT/2000 with the ThreadFiber package



Multithreading Models

4. Two-level Model, a popular variation on n:m model
   - Similar to n:m, except that it allows a user thread to be bound to a kernel thread
   - Examples
     - IRIX
     - HP-UX
     - Tru64 UNIX
     - Solaris 8 and earlier



Outline

- Overview
- Multicore programming
- Multithreading models
- **Thread Libraries**
  - Three main thread libraries
- Threading Issues
- OS examples for thread
- Threading Scheduling

Thread Libraries

中国科学技术大学 University of Science and Technology of China

- A thread library
  provides an API for creating and managing threads.
- Two primary ways
  1. To provide a library entirely in user space with no kernel support
  2. To implement a kernel-level library supported directly by the OS

Library	Code & data	API	Invoking method inside API
User-level	Entirely in user space	User space	A local function call
Kernel-level	Kernel space	User space	System call

- **Two general strategies**
  - **Asynchronous threading**
    - threads are independent with little data sharing
  - **Synchronous threading or fork-join strategy**
    - parent waits until all children have joined
    - involves significant data sharing among threads

Thread Libraries

- Three main thead libraries
  1. POSIX Pthreads
  2. Win32 threads
  3. Java threads

- Example: a multithreaded program
  - Summation:

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

1 Pthreads

- Pthreads
  - A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
  - API specifies behavior of the thread library, implementation is up to development of the library
  - CommoninUNIX OSes (Solaris, Linux, Mac OS X)

Multithreaded C program using the Pthreads API


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

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

/* 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);
}

int main(int argc, char *argv[]) {
    pthread_t tid; /* the thread identifier */
    pthread_attr_t attr; /* set of attributes for the thread */
```

Multithreaded C program using the Pthreads API


```
    if (argc != 2) {
        fprintf(stderr, "usage: a.out \n" );
        return-1;
    }

    if (atoi(argv[1]) < 0) {
        fprintf(stderr, "Argument %d must be non-negative\n" ,atoi(argv[1]));
        return-1;
    }

    pthread_attr_init(&attr); /* get the default attributes */
    pthread_create(&tid, &attr, runner, argv[1]); /* create the thread */
    pthread_join(tid, NULL); /* now wait for the thread to exit */

    printf( "sum = %d\n" , sum);
}
```

pthread_attr_init

NAME
pthread_attr_init, pthread_attr_destroy - initialise and destroy threads attribute object

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

DESCRIPTION
The function pthread_attr_init() initialises a thread attributes object attr with the default value for all of the individual attributes used by a given implementation.
. . .
The pthread_attr_destroy() function is used to destroy a thread attributes object.

RETURN VALUE
Upon successful completion, both return a value of 0.
Otherwise, an error number is returned to indicate the error.
. . .

pthread_create()

中国科学技术大学
University of Science and Technology of China

NAME
pthread_create - thread creation

SYNOPSIS
```
#include <pthread.h>
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine)(void*), void *arg);
```

DESCRIPTION
The pthread_create() function is used to create a new thread, with attributes specified by attr, within a process. . . . Upon successful completion, pthread_create() stores the ID of the created thread in the location referenced by thread.
The thread is created executing start_routine with arg as its sole argument. . . .
. . .
If pthread_create() fails, no new thread is created and the contents of the location referenced by thread are undefined.

RETURN VALUE
If successful, the pthread_create() function returns zero.
Otherwise, an error number is returned to indicate the error.

pthread_join

中国科学技术大学
University of Science and Technology of China

NAME
pthread_join - wait for thread termination

SYNOPSIS
#include <pthread.h>
int **pthread_join**(pthread_t thread, void **value_ptr);

DESCRIPTION
The pthread_join() function suspends execution of the calling thread until the target thread terminates, unless the target thread has already terminated. . . .
The results of multiple simultaneous calls to pthread_join() specifying the same target thread are undefined. . . .

RETURN VALUE
If successful, the pthread_join() function returns zero.
Otherwise, an error number is returned to indicate the error.
. . .

pthread_exit

NAME
pthread_exit - thread termination

SYNOPSIS
#include <pthread.h>
void **pthread_exit**(void *value_ptr);

DESCRIPTION
The pthread_exit() function terminates the calling thread and makes the value value_ptr available to any successful join with the terminating thread. . . .
. . .

RETURN VALUE
The pthread_exit() function cannot return to its caller.

2 Win32 Threads Example

- Similar to the Pthreads technique.
- Multithreaded C program using the Win32 API

```
#include <stdio.h>
#include <windows.h>
DWORD Sum;/* data is shared by the thread(s) */

/* the thread runs in this separate function */
DWORD WINAPI Summation(PVOID Param){
    DWORD Upper =*(DWORD*)Param;
    for (DWORD i = 0; i <= Upper; i++)
        Sum +=i;
    return 0;
}

int main(int argc, char *argv[]){
    DWORD ThreadId;
    HANDLE ThreadHandle;
    int Param;
    // do some basic error checking
```

2 Win32 Threads Example

```
if (argc != 2){
   fprintf(stderr,"An integer parameter is required\n");   return-1;
}

Param = atoi(argv[1]);
if (Param < 0){
   fprintf(stderr, "an integer >= 0 is required \n");    return-1;
}

// create the thread
ThreadHandle = CreateThread(NULL, //default security attribute
                            0, //default stack size
                            Summation, //thread function
                            &Param, //parameter to thread function
                            0, //default creation flags
                            &ThreadId);

if (ThreadHandle != NULL){
    WaitForSingleObject(ThreadHandle, INFINITE);
    CloseHandle(ThreadHandle);

    printf("sum = %d\n",Sum);
}
}
```

3 Java Threads

- Java Threads
  - **Threads are the fundamental model** for program execution.
  - Java threads may be created by:
    - **Extending Thread class**
      to create a new class that is derived from the Thread class and override its run() method.
    - **Implementing the Runnable interface**

Java Thread Example

中国科学技术大学 University of Science and Technology of China

```
class Sum {
  private int sum;

  public int get() {
    return sum;
  }

  public void set(int sum) {
    this.sum = sum;
  }
}
```

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

  public Summation(int upper, Sum sumValue) {
    if (upper < 0) throw new
IllegalArgumentException();
    this.upper = upper;
    this.sumValue = sumValue;
  }

  public void run() {
    int sum = 0;
    for (int i = 0; i <= upper; i++)
      sum += i;
    sumValue.set(sum);
  }
}
```

Java Thread Example

```
public class Driver {
  public static void main(String[] args) {
    if (args.length != 1) {
      System.err.println( "Usage Driver <integer>" );
      System.exit(0);
    }

    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) { }

    System.out.println( "The sum of" + upper + " is " + sumObject.get());
  }
}
```

Outline

- Overview
- Multicore programming
- Multithreading models
- Thread Libraries
- **Threading Issues**
  - To discuss some of the issues to consider in designing multi threaded programs
- OS examples for thread
- Threading Scheduling

Threading Issues

- **Implicit threading**：(VS. explicit threading)
  - Developing multithreaded application is **hard, error-prone and time-consuming**
  - How to **transfer** the creation and management of threading from APPs developers **to compilers and run-time LIBs**.
  - Example：
    - Fork-join model; OpenMP; Grand Central Dispatch (大中央调度，GCD); Intel Thread Building Blocks (Intel的线程构建模块，TBB); …
- **Thread Pools**:
  - At startup, create a number of threads in a pool where they await work
  - Advantages:
    - Usually slightly **faster** to service a request with an existing thread than create a new thread
    - Allows the number of threads in the APPs to **be bound to the size of the pool**
    - Separates the task from the mechanics of creating the task, **flexibility↑**

Threading Issues

- **Semantics of fork() and exec() system calls**
  - Does fork() duplicate only the calling thread or all threads?
  - Some UNIX system have chosen to have two versions
  - Which one version to use? Depend on the APP.
- **Signal Handling** (ignore, or by yourself)
- **Thread cancellation** (线程撤销)
  - Terminating a thread before it has finished
  - **Two general approaches**:
    - **Asynchronous cancellation** terminates the target thread immediately
    - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled

Threading Issues

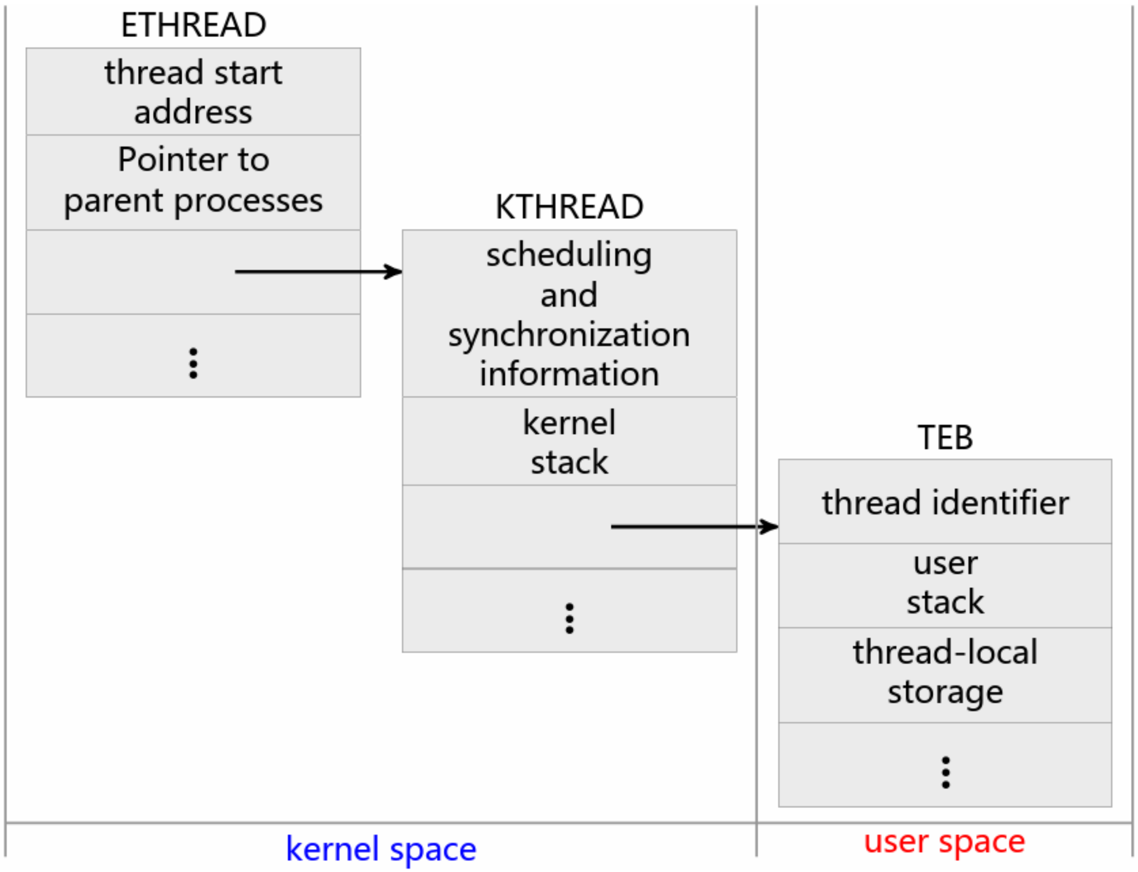
- **Thread-Local Storage**
  - Allows each thread to have its own copy of certain data
  - similar to static data, but unique to each thread
- **Scheduler Activations** (调度器激活)
  - Both n:m and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
  - Scheduler activations provide upcalls- a communication mechanism from the kernel to the thread library
  - This communication allows an application to maintain the correct number kernel threads

Outline

- Overview
- Multicore programming
- Multithreading models
- Thread Libraries
- Threading Issues
- **OS examples for thread**
  - Windows Threads
  - Linux Threads
  - Java Threads
- Threading Scheduling

Windows Threads

- The primary data structures of a thread include **ETHREAD** (executive thread block), **KTHREAD**(kernel thread block) ; **TEB**(thread environment block)



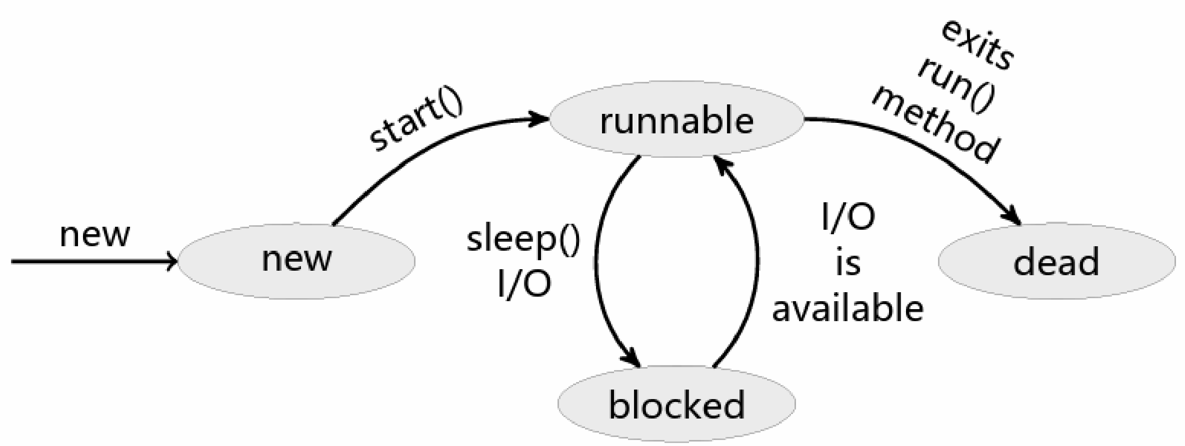
Linux Threads

- Linux refers to them as tasks rather than threads
- Thread creation is done through clone() system call
- clone() allows **a child task to share the address space of the parent task (process)**
- clone() VS. fork()

Flag	Meaning
CLONE_FS	File-system information is shared
CLONE_VM	The same memory space is shared
CLONE_SIGHAND	Signal handlers are shared
CLONE_FILES	The set of open files is shared

Java Threads

- Java在**语言级**提供线程创建和管理支持功能
  - Java threads are managed by the JVM, not user-level library or kernel
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface Java



thead States

Outline

- Overview
- Multicore programming
- Multithreading models
- Thread Libraries
- Threading Issues
- OS examples for thread
- Threading Scheduling

Threading Scheduling

- **user-level thread** VS. **kernel-level thread (or LWP)**
- **Local Scheduling**– How the **threads library** decides which thread to put onto an available LWP
  - many-to-one, many-to-many models
  - process-contention scope, PCS
- **Global Scheduling**– How the **kernel** decides which kernel thread to run next
  - many-to-one, many-to-many & one-to-one models
  - system-contention scope, SCS

Pthread Scheduling API

- POSIX Pthread API **allows specifying either PCS** or SCS during thread creation
  - PTHREAD_SCOPE_PROCESS, many-to-many
  - PTHREAD_SCOPE_SYSTEM, one-to-one
    - create and **bind** an LWP for each user-level thread
- example

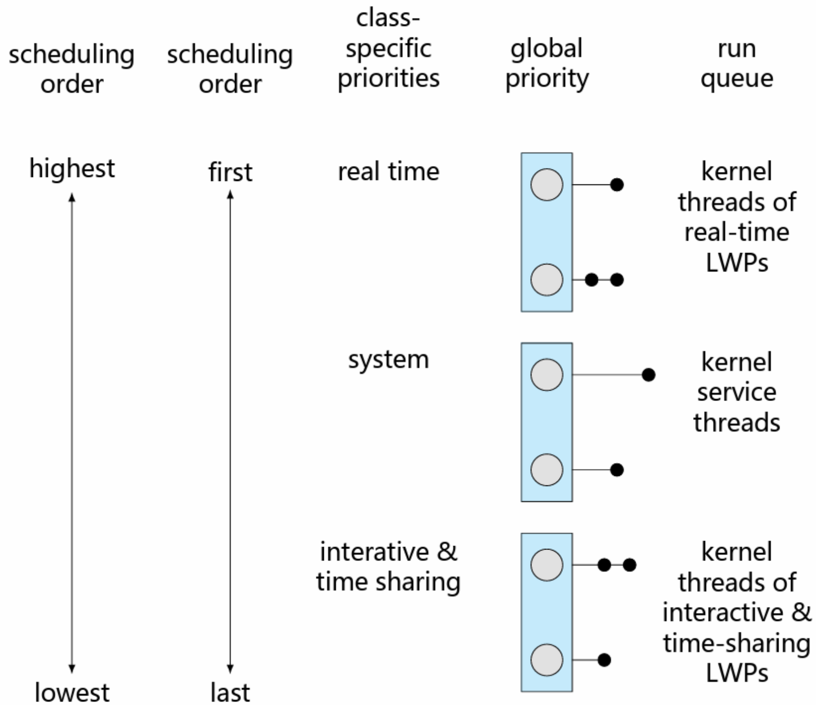
```
#include <pthread.h>
#include <stdio.h>
#define NUM THREADS 5

int main(int argc, char *argv[]) {
    int i;
    pthread_t tid[NUM THREADS];
    pthread_attr_t attr;
    pthread_attr_init(&attr); /* get the default attributes */

    /* set the scheduling algorithm to PROCESS or SYSTEM */
    pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
```

Pthread Scheduling API

```
    /* set the scheduling policy- FIFO, RT, or OTHER */
    pthread_attr_setschedpolicy(&attr, SCHED_OTHER);

    for (i = 0; i < NUM_THREADS; i++) /* create the threads */
        pthread_create(&tid[i],&attr,runner,NULL);

    for (i = 0; i < NUM THREADS; i++) /* now join on each thread */
        pthread_join(tid[i], NULL);
}

/* Each thread will begin control in this function */
void *runner(void *param) {
    printf( "I am a thread\n" );

    pthread_exit(0);
}
```

Solaris scheduling

- Solaris: **priority-based** thread scheduling
- **4 classes of scheduling**, in order of priority. Within each class there are **different priorities** and **different scheduling algorithms**.
  - **Real time**
  - **System** (do not change the priority)
  - **Time sharing** (default, with a multilevel feedback queue)
  - **Interactive**, the same as time sharing, but higher priority



Solaris scheduling

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Solaris Dispatch Table

Windows XP scheduling

- Dispatcher: **priority-based**, **preemptive** scheduling algorithm uses a **32-level priority** scheme to determine the order of thread execution
  - 0: idle thread
  - 1~15: variable classes priorities
  - 16~31: real-time class
  - One queue for each priority

	real–time	high	above normal	normal	below normal	idle priority
time-cribical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Windows XP Priorities (**policy classes**, **relative priority**)

小结

- Overview
- Multicore programming
- Multithreading models
- Thread Libraries
- Threading Issues
- OS examples for thread
- Threading Scheduling

Q & A