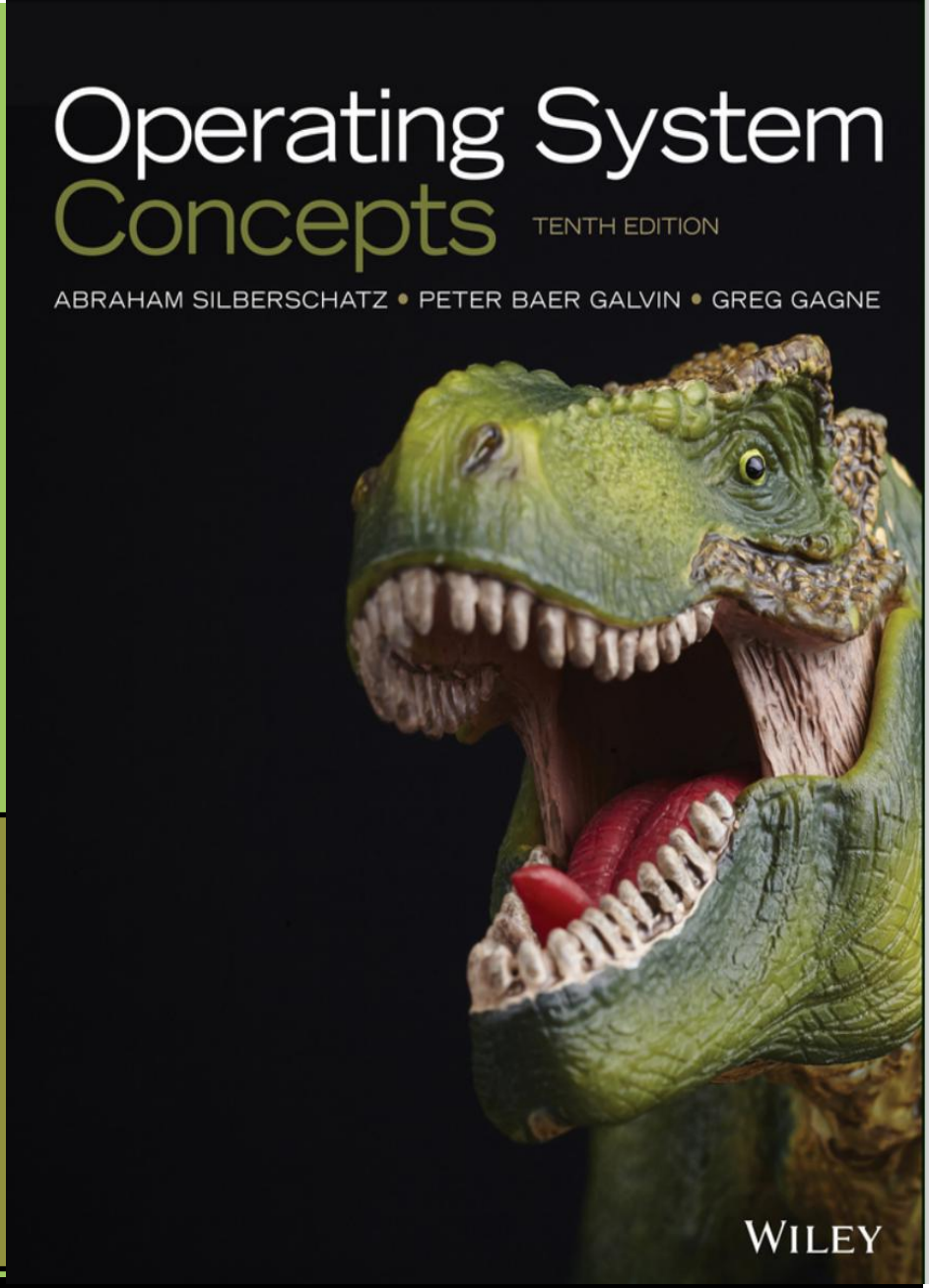
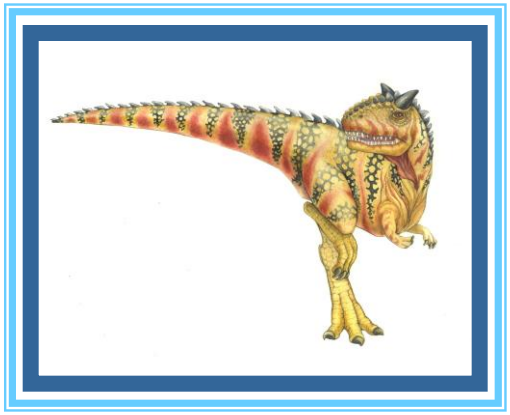
Operation Systems

Dr. A. Taghinezhad



Website: ataghinezhad.github.io, Email:a0taghinezhad@gmail.com

فصل ۴: رشته‌ها و همزمانی

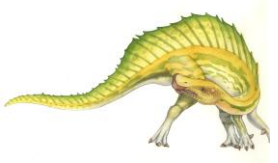


Outline

- برنامه نویسی چند هسته ای (Multicore Programming)
- مدل های چند رشته ای (Multithreading Models)
- کتابخانه های ترد (Thread Libraries)
- رشته بندی ضمنی (Implicit Threading)
- مسائل مربوط به چند رشته ای(Threading Issues)
- نمونه های سیستم عامل(Operating System Examples)

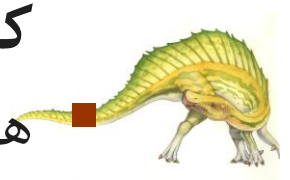
Objectives

- اجزای اصلی یک ترد را شناسایی کنید و تردها را با فرایندها مقایسه نمایید.
- توضیح دهید که طراحی برنامه های چند رشته ای چه مزایا و چالش هایی به همراه دارد.
- رویکردهای مختلف برای رشته بندی ضمنی از جمله thread pool ها، fork-join و Grand Central Dispatch را شرح دهید.
- نحوه نمایش تردها در سیستم عامل های ویندوز و لینوکس را توضیح دهید.
- با استفاده از API های رشته بندیPthreads ، جاوا و ویندوز، نحوه طراحی برنامه های چند رشته ای را شرح دهید.

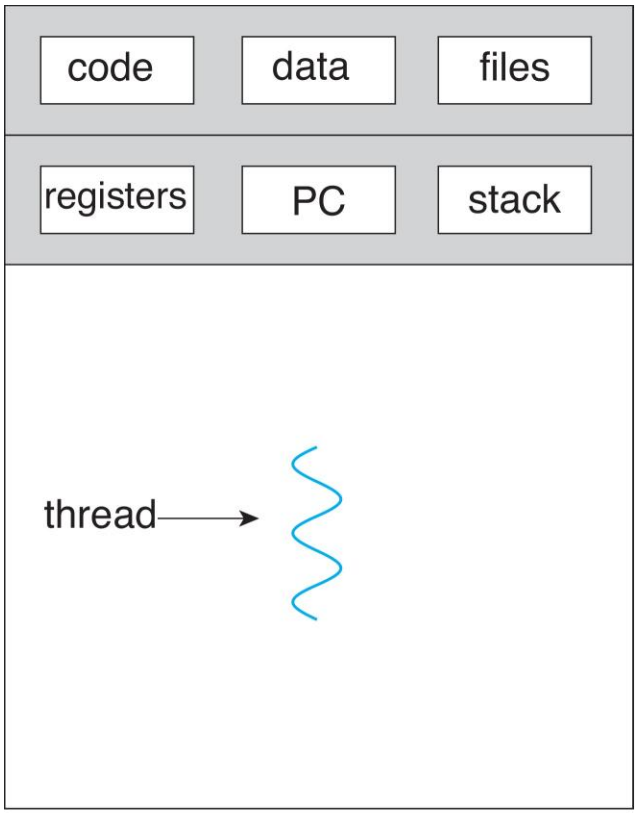


Motivation

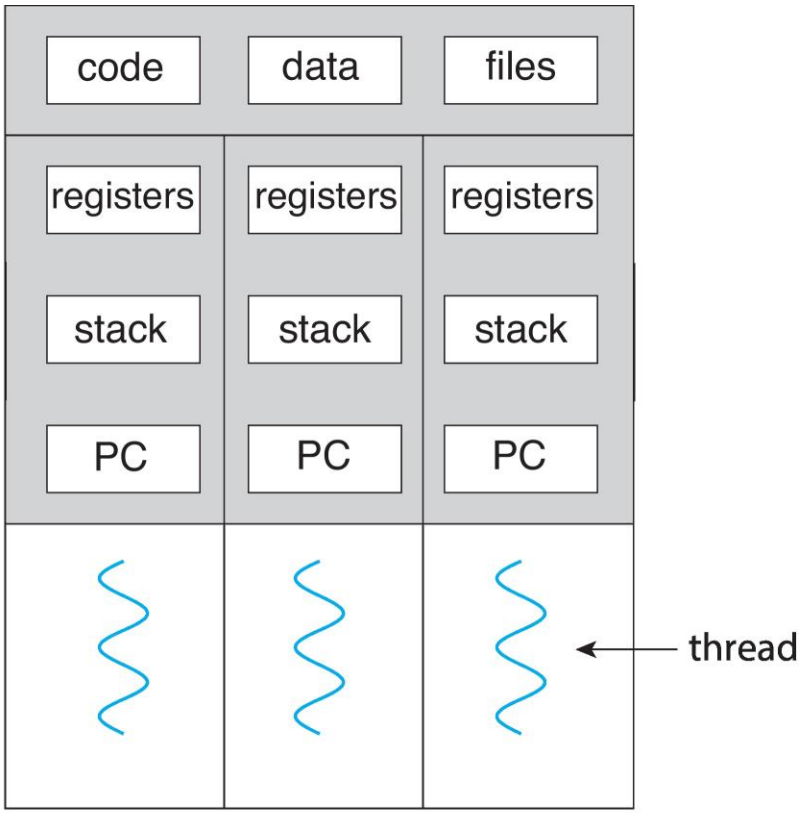
- اکثر برنامه های کاربردی مدرن چند رشته ای هستند.
- تردها درون یک برنامه اجرا می شوند.
- وظایف متعدد یک برنامه را می توان با تردهای جداگانه اجرا کرد.
  - به روز رسانی رابط کاربری
  - دریافت داده ها
  - بررسی املای متن
  - پاسخ به درخواست شبکه
- ایجاد فرایند یک کار سنگین است، در حالی که ایجاد ترد سبک وزن است.
- استفاده از تردها می تواند باعث **ساده سازی کد** و افزایش **کارایی** برنامه شود.
- هسته های سیستم عامل به طور کلی چند رشته ای هستند.



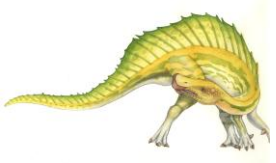
Single and Multithreaded Processes



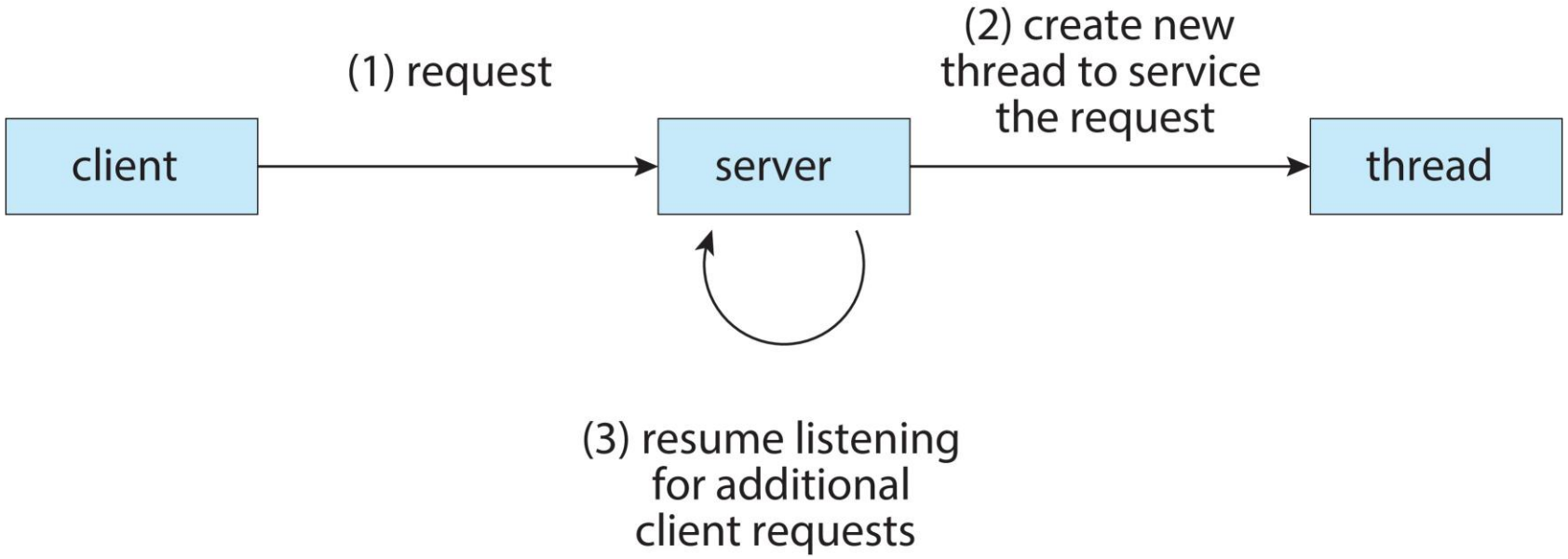
single-threaded process



multithreaded process



Multithreaded Server Architecture

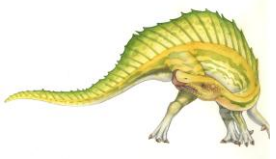


مزایای چند رشته ای بودن

- اکثر برنامه های کاربردی مدرن چند رشته ای هستند.
- تردها درون یک برنامه اجرا می شوند.
- وظایف متعدد یک برنامه را می توان با تردهای جداگانه اجرا کرد.
  - به روز رسانی رابط کاربری
  - دریافت داده ها
  - بررسی املای متن
  - پاسخ به درخواست شبکه
- ایجاد فرایند یک کار سنگین است، در حالی که ایجاد ترد سبک وزن است.
- استفاده از تردها می تواند باعث ساده سازی کد و افزایش کارایی برنامه شود.
- هسته های سیستم عامل به طور کلی چند رشته ای هستند.

Multicore Programming

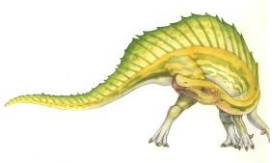
- سیستم های چند هسته ای یا چند پردازنده برنامه نویسان را با چالش هایی مواجه می کنند که برخی از آنها عبارتند از:
  - تقسیم فعالیت‌ها
  - توازن
  - تقسیم‌بندی داده ها
  - وابستگی داده‌ها
  - تست و اشکال‌زدایی



Multicore Programming

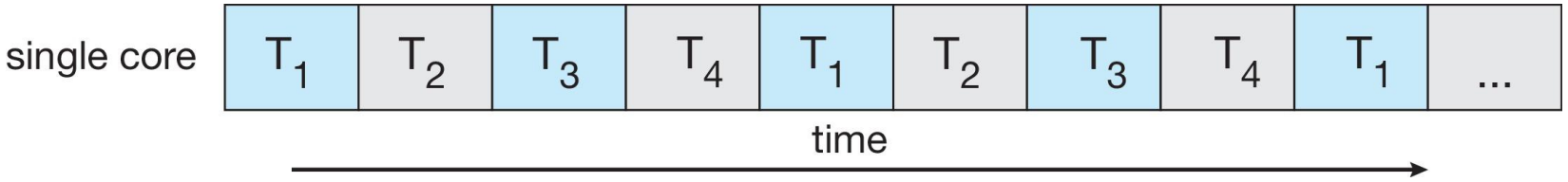
- **موازی کاری در مقابل همزمانی (Parallelism vs Concurrency):**

- **موازی کاری (Parallelism):** این مفهوم به توانایی یک سیستم برای انجام بیش از یک کار به طور همزمان اشاره دارد.
- **همزمانی (Concurrency):** این مفهوم به توانایی یک سیستم برای پیشبرد بیش از یک کار اشاره دارد. حتی در یک پردازنده یا هسته واحد، زمانبندی کننده سیستم می تواند همزمانی را فراهم کند.

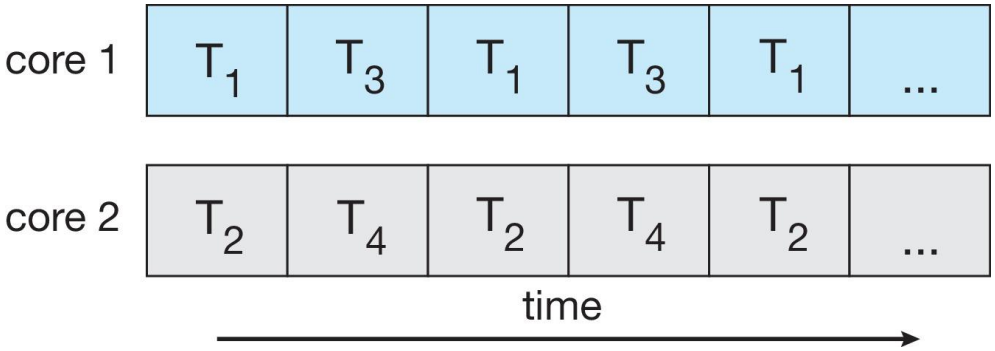


Concurrency vs. Parallelism

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



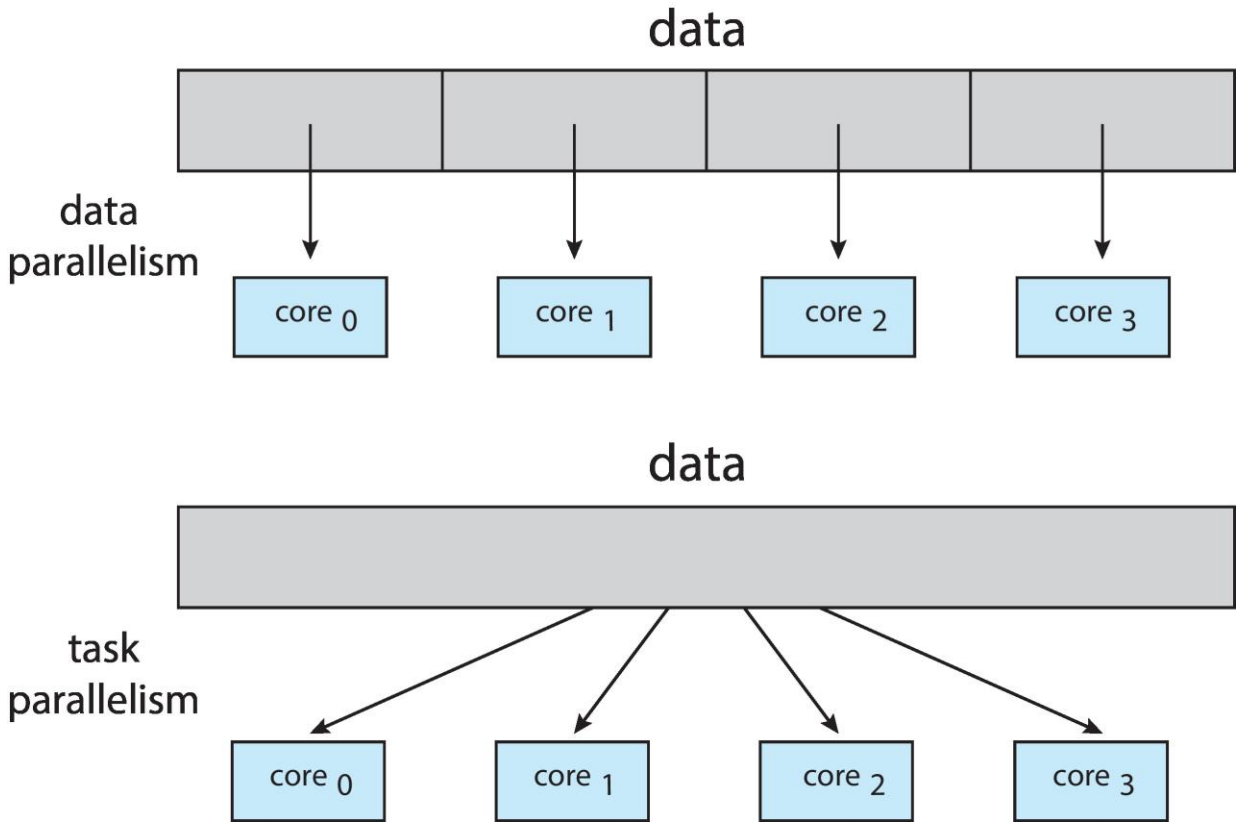
- **Parallelism on a multi-core system:**



Multicore Programming

- **موازی کاری داده (Data parallelism):** توزیع زیرمجموعه هایی از داده های مشابه در هسته های مختلف، به طوری که عملیات یکسانی روی هر مجموعه داده انجام شود.
- **موازی کاری وظیفه (Task parallelism):** توزیع تردها در هسته های مختلف، به طوری که هر ترد یک عملیات منحصر به فرد را انجام دهد.

Data and Task Parallelism



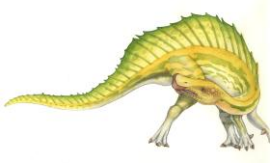
قانون امدل

- این قانون، سود بالقوه ناشی از اضافه کردن هسته های اضافی به یک برنامه کاربردی را که دارای اجزای سریال و موازی است، مشخص می کند.

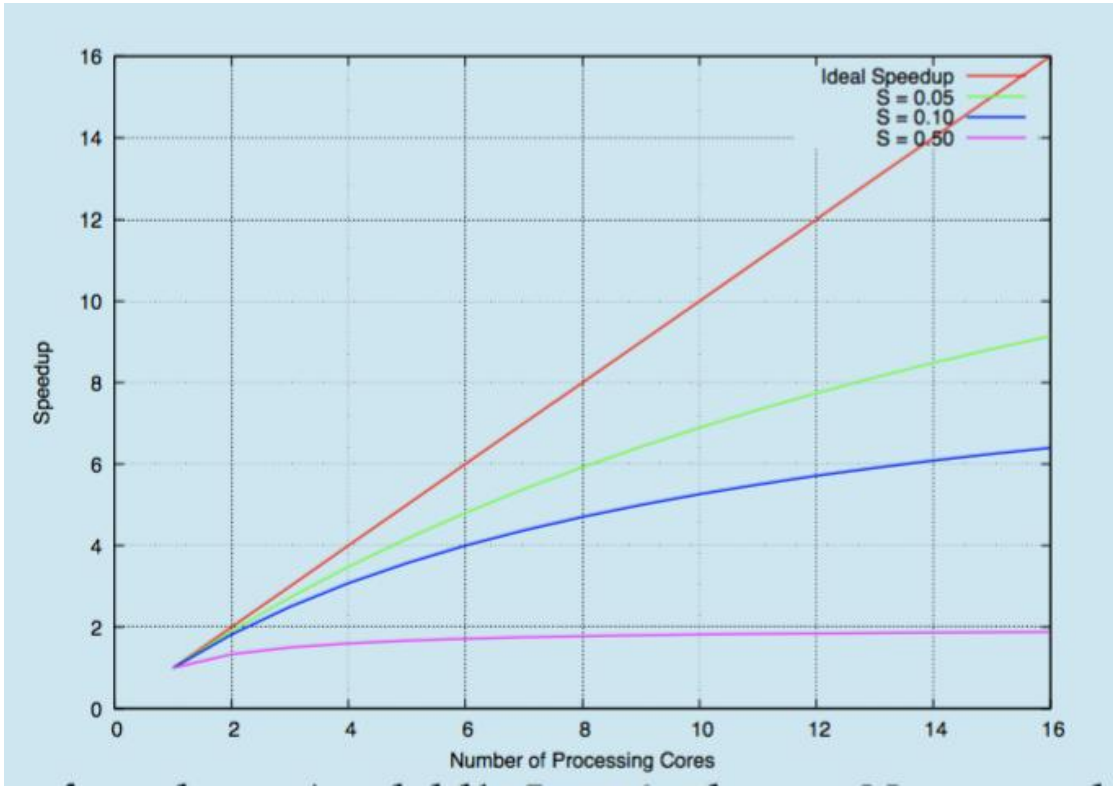
$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

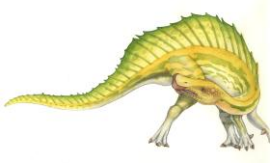
- Sبخش سریال برنامه است.
- Nتعداد هسته های پردازنده است.

- به عبارت دیگر، اگر یک برنامه کاربردی ۷۵ درصد موازی و ۲۵ درصد سریال باشد، انتقال از ۱ هسته به ۲ هسته منجر به افزایش سرعت ۱.۶ برابری می شود.
- با نزدیک شدن N به بی نهایت، افزایش سرعت به ۱ / S نزدیک می شود.
- بخش سریال یک برنامه تاثیر نامتناسبی بر روی بهبود عملکرد ناشی از اضافه کردن هسته‌های اضافی دارد.



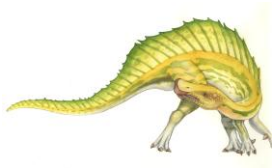
Amdahl's Law


Ideal Speedup
S = 0.05
S = 0.10
S = 0.50
Speedup
Number of Processing Cores
0
2
4
6
8
10
12
14
16

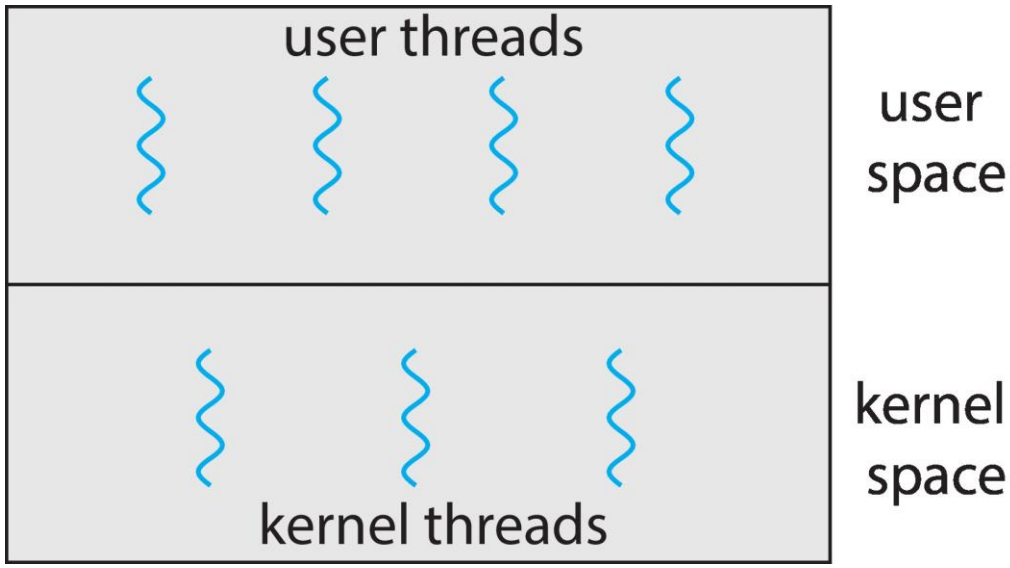


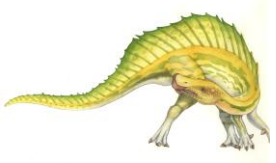
User Threads and Kernel Threads

- **تردهای کاربر (User Threads):** مدیریت این نوع ترد توسط کتابخانه ترد در سطح کاربر انجام می شود.
  - نمونه ها:
    - تردهای POSIX (Pthreads)
    - تردهای ویندوز
    - تردهای جاوا
- **تردهای هسته (Kernel Thread):** این نوع ترد توسط هسته سیستم عامل پشتیبانی می شود.
  - نمونه ها:
    - تقریباً تمام سیستم عامل های عمومی، از جمله:
      - ویندوز – لینوکس-مکینتاش-اندروید



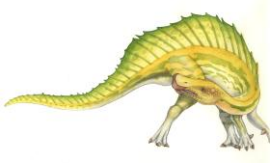
User and Kernel Threads





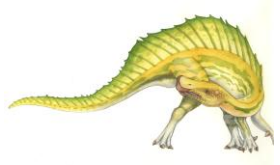
مدل‌های چندرشته‌ای

- Many-to-Oneچند به یک
- One-to-One یکی به یک
- Many-to-Many چند به چند



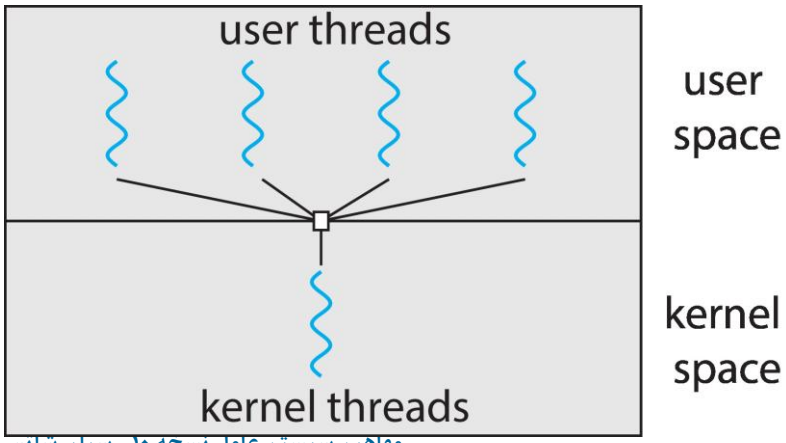
Many-to-One

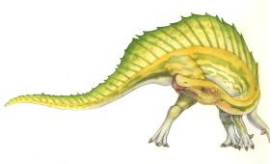
- **تردهای کاربر (User Threads):** مدیریت این نوع ترد توسط کتابخانه ترد در سطح کاربر انجام می شود.
  - نمونه ها:
    - تردهای POSIX (Pthreads)
    - تردهای ویندوز
    - تردهای جاوا
- **تردهای هسته (Kernel Thread):** این نوع ترد توسط هسته سیستم عامل پشتیبانی می شود.
  - نمونه ها:
    - تقریباً تمام سیستم عامل های عمومی، از جمله:
      - ویندوز-لینوکس-مکینتاش-iOS-اندروید



Many-to-One

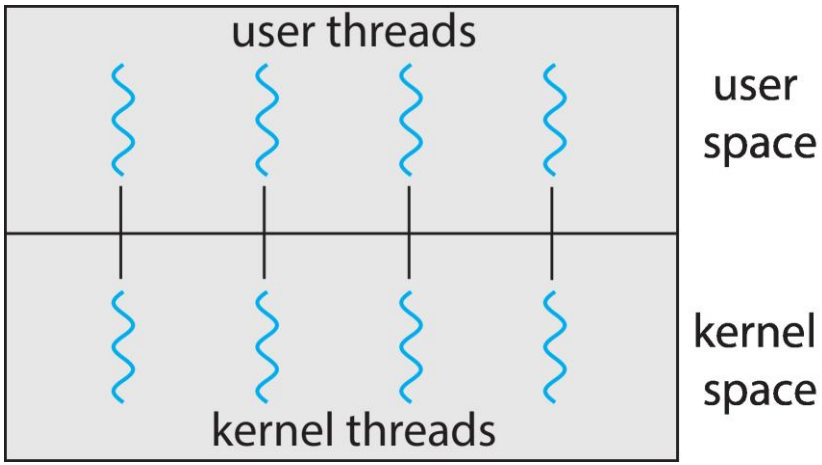
- مدل چند به یک (شکل ۴,۷) رشته‌های زیادی در سطح کاربر را به یک رشته هسته نگاشت می‌کند.
- بدلیل اینکه مدیریت رشته‌ها توسط کتابخانه رشته در فضای کاربر انجام می‌شود، این روش کارآمد است (کتابخانه‌های رشته در بخش ۴,۴ مورد بحث قرار خواهند گرفت).
- با این حال، کل فرایند در صورتی که یک رشته فراخوانی سیستم **مسدود کننده** انجام دهد، مسدود خواهد شد.
- فقط یک رشته می‌تواند به هسته در یک زمان دسترسی پیدا کند، چندین رشته قادر به اجرای موازی در سیستم‌های چند هسته‌ای نیستند.

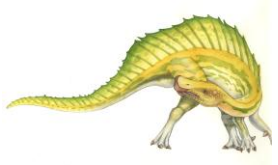




One-to-One

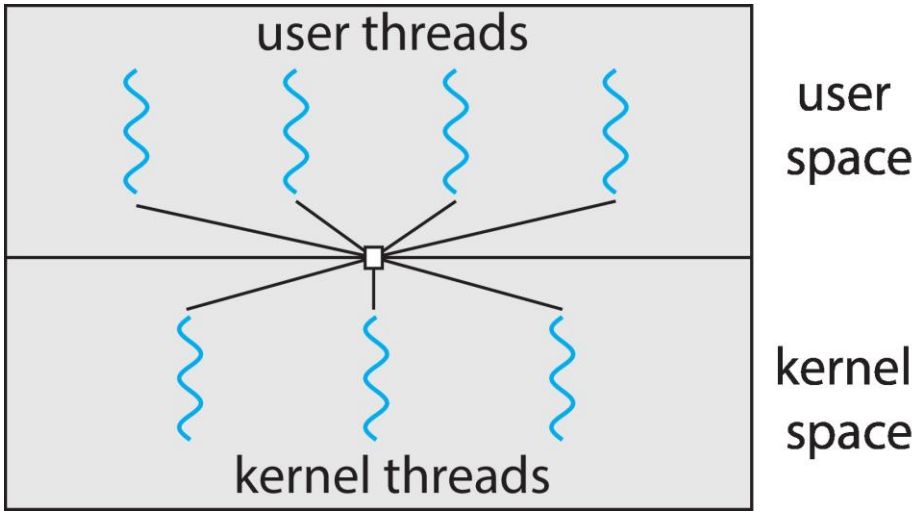
- هر ترد کاربر به یک ترد هسته نگاشت می شود.
- ایجاد یک ترد کاربر، یک ترد هسته ایجاد می کند.
- همزمانی (Concurrency) بیشتری نسبت به مدل چند به یک (Many-to-One) دارد.
- تعداد زیاد تردهای هسته به ازای هر فرایند گاهی اوقات به دلیل سربار (overhead) محدود می شود.
  - نمونه ها:
    - ویندوز
    - لینوکس

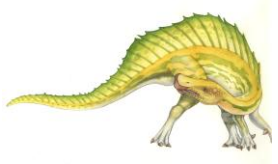




Many-to-Many Model

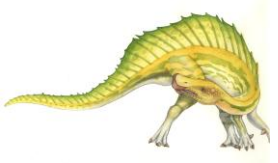
- اجازه می‌دهد **چندین ترد** در سطح کاربر به **چندین ترد** (کمتر یا برابر) در سطح هسته نگاشت شوند.
- این مدل به سیستم عامل اجازه می دهد تا تعداد کافی ترد کاربر ایجاد کند. اما این کار منجر به موازی‌سازی نمی‌شود. زیرا هسته فقط می‌تواند یک رشته **هسته** را در یک زمان زمان‌بندی کند.





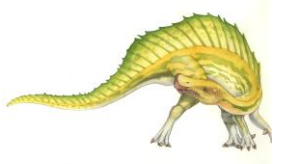
Thread Libraries

- کتابخانه ترد، API ایجاد و مدیریت تردها را برای برنامه نویس فراهم می کند.
- دو روش اصلی برای پیاده سازی وجود دارد:
  - کتابخانه به طور کامل در فضای کاربر
  - کتابخانه سطح هسته که توسط سیستم عامل پشتیبانی می شود



Pthreads

- ممکن است به صورت سطح کاربر یا سطح هسته ارائه شود.
- یک API استاندارد POSIX (IEEE 1003.1c) است برای ایجاد و همگام سازی ترد
  - API رفتار کتابخانه ترد را مشخص می کند، پیاده سازی به توسعه کتابخانه بستگی دارد.
  - این نوع کتابخانه در سیستم عامل های یونیکس (لینوکس و مکینتاش) رایج است.
  - Pthreads به‌صورت مستقیم در ویندوز پشتیبانی نمی‌شوند ولی یک برنامه شخص‌ثالث برای آن وجود دارد.



Pthreads Example

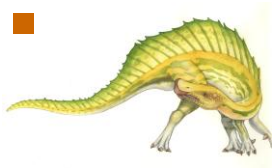
```
#include <pthread.h>
#include <stdio.h>

#include <stdlib.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

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

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

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

 /* The thread will execute in this function */
 void *runner(void *param)
 {
    int i, upper = atoi(param);
    sum = 0;

    for (i = 1; i <= upper; i++)
       sum += i;

    pthread_exit(0);
 }
```

Java Threads

- تردهای جاوا توسط JVM مدیریت می شوند.
  - به طور معمول با استفاده از مدل تردهایی که توسط سیستم عامل ارائه می شود، اجرا می شوند.
  - تردهای جاوا ممکن است با موارد زیر ایجاد شوند:
    - توسعه (Extend) دادن کلاس Thread
    - پیاده سازی رابط Runnable

```
public interface Runnable
{
    public abstract void run();
}
```

- رویه استاندارد، پیاده سازی رابط Runnable است.



Java Threads

Implementing Runnable interface:

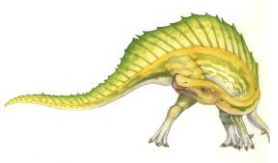
```
class Task implements Runnable
{
   public void run() {
      System.out.println("I am a thread.");
   }
}
```

Creating a thread:

```
Thread worker = new Thread(new Task());
worker.start();
```

Waiting on a thread:

```
try {
   worker.join();
}
catch (InterruptedException ie) { }
```



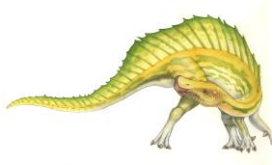
Java Executor Framework

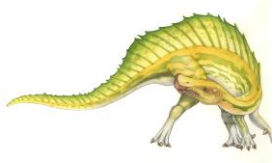
- جاوا به جای ایجاد صریح رشته‌ها، اجازه می‌دهد تا در اطراف رابط Executorایجاد رشته کند:

```
public interface Executor
{
   void execute(Runnable command);
}
```

- این رابط به این شکل اجرا می‌شود.

```
Executor service = new Executor;
service.execute(new Task());
```





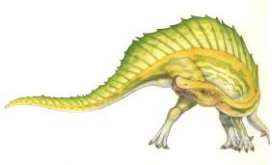
Java Executor Framework

- Rather than explicitly creating threads, Java also allows thread creation around the Executor interface:

```
public interface Executor
{
   void execute(Runnable command);
}
```

- The Executor is used as follows:

```
Executor service = new Executor;
service.execute(new Task());
```

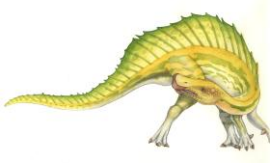


Java Executor Framework

```
import java.util.concurrent.*;

class Summation implements Callable<Integer>
{
   private int upper;
   public Summation(int upper) {
      this.upper = upper;
   }

   /* The thread will execute in this method */
   public Integer call() {
      int sum = 0;
      for (int i = 1; i <= upper; i++)
         sum += i;

      return new Integer(sum);
   }
}
```

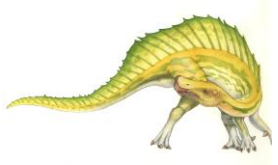
Java Executor Framework (Cont.)

```
public class Driver
{
 public static void main(String[] args) {
    int upper = Integer.parseInt(args[0]);

    ExecutorService pool = Executors.newSingleThreadExecutor();
    Future<Integer> result = pool.submit(new Summation(upper));

    try {
        System.out.println("sum = " + result.get());
    } catch (InterruptedException | ExecutionException ie) { }
  }
}
```



Implicit Threading

- با افزایش تعداد تردها، از محبوبیت بیشتری برخوردار می شود، درستی برنامه با تردهای صریح (Explicit Threads) دشوارتر است.
- ایجاد و مدیریت تردها توسط کامپایلرها و کتابخانه های زمان اجرا به جای برنامه نویسان انجام می شود.
- پنج روش بررسی شده:
  - Thread Poolsاستخر ترد
  - Fork-Join
  - OpenMP
  - Grand Central Dispatch
  - Intel Threading Building Blocks



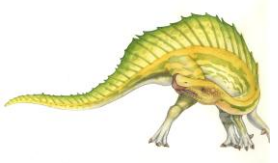
Thread Pools

- تعدادی ترد در یک استخر ایجاد کنید که در آنجا منتظر کار هستند.
- مزایا:
  - سرویس دهی به یک درخواست با یک ترد موجود معمولاً کمی سریعتر از ایجاد یک ترد جدید است.
  - به برنامه (های) کاربردی اجازه می دهد تا تعداد تردها را به اندازه استخر محدود کند.
  - جدا کردن وظیفه ای که باید انجام شود از مکانیک ایجاد وظیفه، استراتژی های مختلفی را برای اجرای وظیفه امکان پذیر می کند.
    - به عنوان مثال، وظایف می توانند به صورت دوره ای برای اجرا برنامه ریزی شوند.
- APIویندوز از استخرهای ترد پشتیبانی می کند.

```
DWORD WINAPI PoolFunction(AVOID Param) {
    /*
    * this function runs as a separate thread.
    */
}
```

Java Thread Pools

- Three factory methods for creating thread pools in Executors class:
  - `static ExecutorService newSingleThreadExecutor()`
  - `static ExecutorService newFixedThreadPool(int size)`
  - `static ExecutorService newCachedThreadPool()`

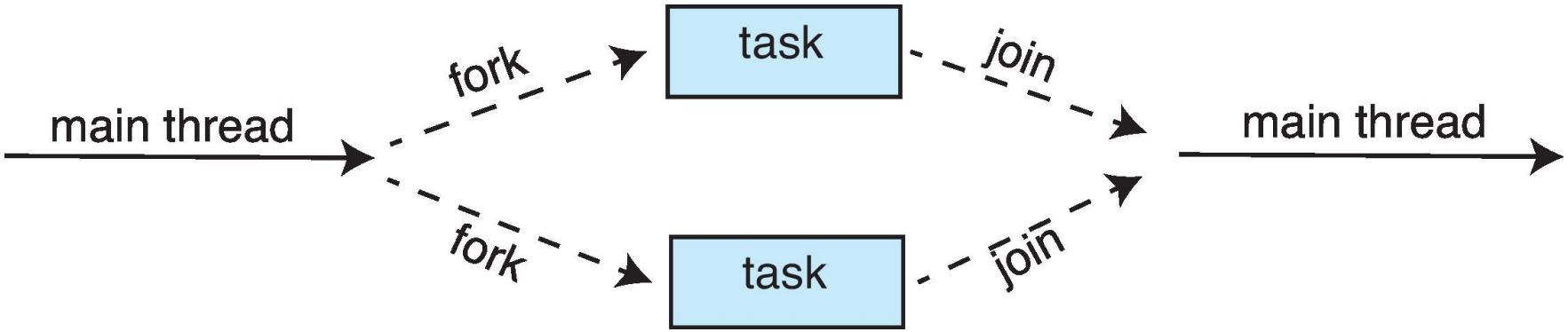


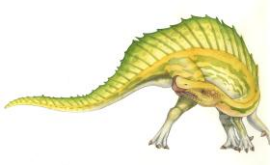
Java Thread Pools (Cont.)

```
import java.util.concurrent.*;

public class ThreadPoolExample
{
public static void main(String[] args) {
   int numTasks = Integer.parseInt(args[0].trim());

   /* Create the thread pool */
   ExecutorService pool = Executors.newCachedThreadPool();

   /* Run each task using a thread in the pool */
   for (int i = 0; i < numTasks; i++)
      pool.execute(new Task());

   /* Shut down the pool once all threads have completed */
   pool.shutdown();
}
```

Fork-Join Parallelism

- Multiple threads (tasks) are **forked**, and then **joined**.



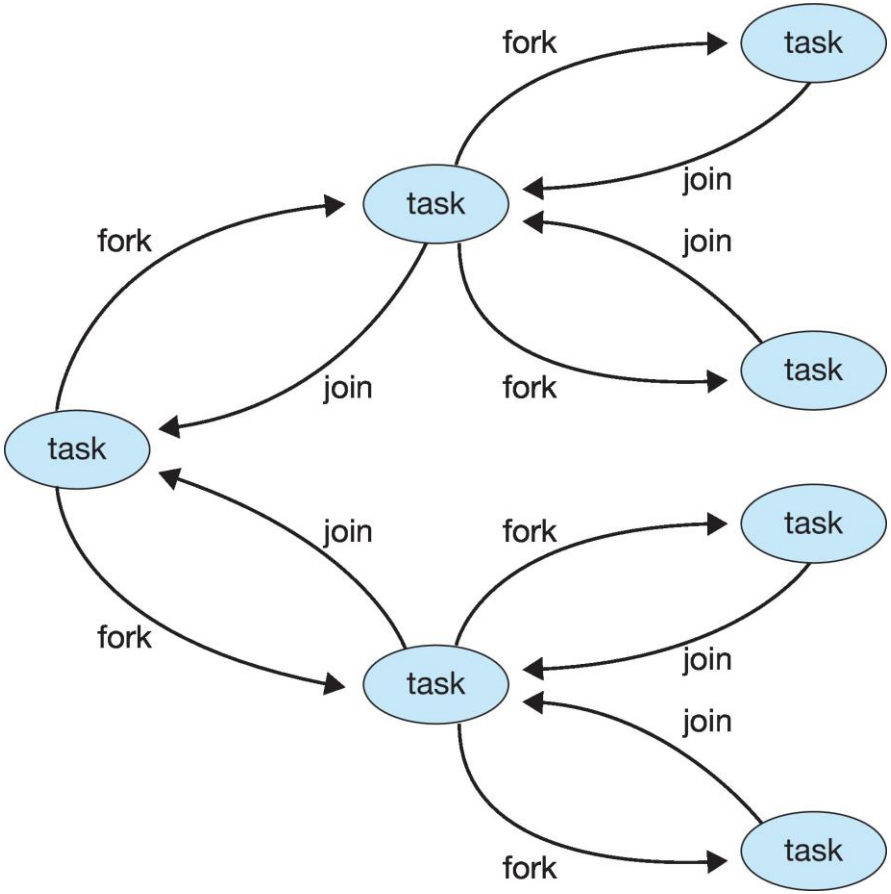


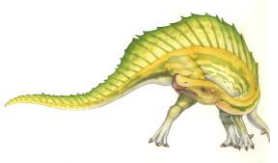
Fork-Join Parallelism

- General algorithm for fork-join strategy:

```
Task(problem)
   if problem is small enough
     solve the problem directly
   else
     subtask1 = fork(new Task(subset of problem)
     subtask2 = fork(new Task(subset of problem)

     result1 = join(subtask1)
     result2 = join(subtask2)

     return combined results
```

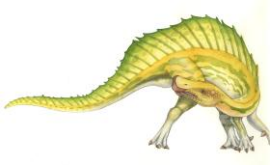
Fork-Join Parallelism





Fork-Join Parallelism in Java

```
ForkJoinPool pool = new ForkJoinPool();
// array contains the integers to be summed
int[] array = new int[SIZE];

SumTask task = new SumTask(0, SIZE - 1, array);
int sum = pool.invoke(task);
```

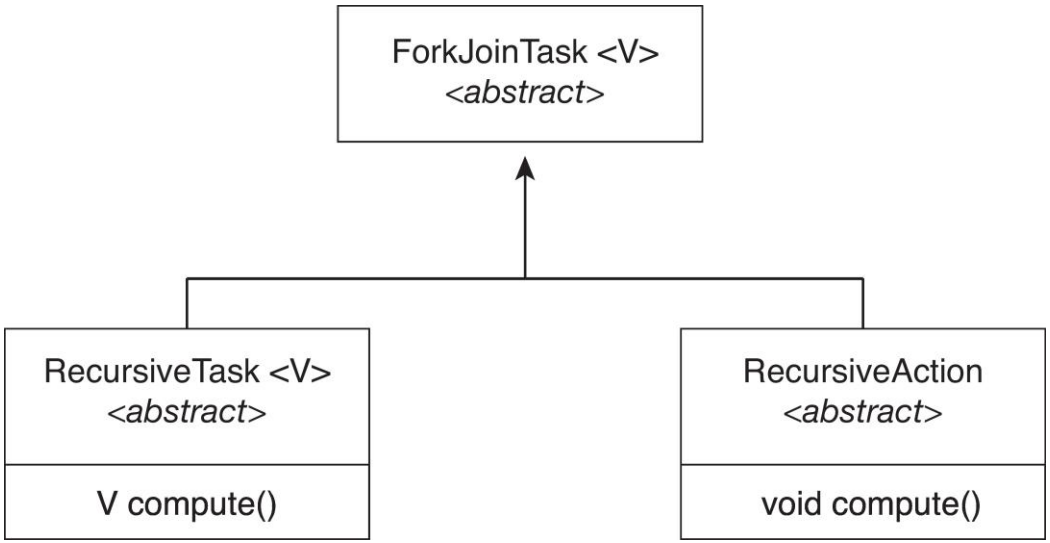


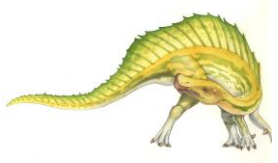
Fork-Join Parallelism in Java

```
import java.util.concurrent.*;

public class SumTask extends RecursiveTask<Integer>
{
   static final int THRESHOLD = 1000;

   private int begin;
   private int end;
   private int[] array;

   public SumTask(int begin, int end, int[] array) {
      this.begin = begin;
      this.end = end;
      this.array = array;
   }

   protected Integer compute() {
      if (end - begin < THRESHOLD) {
         int sum = 0;
         for (int i = begin; i <= end; i++)
            sum += array[i];

         return sum;
      }
      else {
         int mid = (begin + end) / 2;

         SumTask leftTask = new SumTask(begin, mid, array);
         SumTask rightTask = new SumTask(mid + 1, end, array);

         leftTask.fork();
         rightTask.fork();

         return rightTask.join() + leftTask.join();
      }
   }
}
```

Fork-Join Parallelism in Java

- The **`ForkJoinTask`** is an abstract base class
- **`RecursiveTask`** and **`RecursiveAction`** classes extend **`ForkJoinTask`**
- **`RecursiveTask`** returns a result (via the return value from the **`compute()`** method)
- **`RecursiveAction`** does not return a result





Thread Cancellation

- **خاتمه دادن به یک رشته قبل از اتمام آن**
- رشته‌ای که باید لغو شود، رشته هدف است.
- دو رویکرد کلی:
  - لغو ناهمزمان رشته هدف را بلافاصله خاتمه می دهد.
  - لغو به تعویق افتاده به رشته هدف اجازه می دهد تا به صورت دوره ای بررسی کند که آیا باید لغو شود.
- کد Pthread برای ساختن و لغو یک نخ

```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

  . . .

/* cancel the thread */
pthread_cancel(tid);

/* wait for the thread to terminate */
pthread_join(tid,NULL);
```

Thread Cancellation (Cont.)

- فراخوانی لغو رشته، لغو را درخواست می کند، اما لغو واقعی به وضعیت رشته بستگی دارد.

Mode	State	Type
Off	Disabled	–
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- اگر لغو برای رشته غیرفعال باشد، لغو تا زمانی که رشته آن را فعال کند معلق باقی می ماند.
- نوع پیش فرض به تعویق افتاده است.
- لغو فقط زمانی رخ می دهد که رشته به نقطه لغو برسد.
- یعنی pthread_testcancel()
- سپس هندلر پاکسازی فراخوانده می شود.
- در سیستم های لینوکس، لغو رشته از طریق سیگنال ها مدیریت می شود.

Thread Cancellation in Java

- لغو به تعویق افتاده از روش interrupt() استفاده می کند که وضعیت وقفه یک رشته را تنظیم می کند.

```
Thread worker;

  . . .

/* set the interruption status of the thread */
worker.interrupt()
```

- سپس یک رشته می تواند بررسی کند که آیا قطع شده است یا خیر:

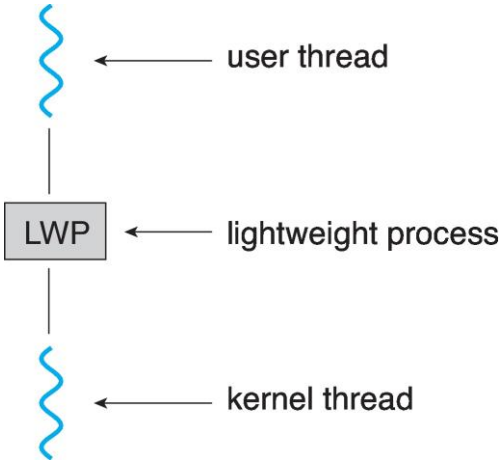
```
while (!Thread.currentThread().isInterrupted()) {
    . . .
}
```

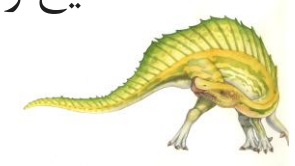
Thread-Local Storage

- ذخیره سازی محلی رشته (TLS) به هر رشته اجازه می دهد تا یک کپی از داده های خود داشته باشد.
- زمانی مفید است که روی فرآیند ایجاد رشته کنترلی نداشته باشید (یعنی زمانی که از استخر رشته استفاده می کنید).
- با متغیرهای محلی متفاوت است.
- متغیرهای محلی فقط در طول فراخوانی یک تابع خاص قابل مشاهده هستند.
- TLSدر سراسر فراخوانی توابع قابل مشاهده است.
- مشابه داده های استاتیک
- TLSبرای هر رشته منحصر به فرد است.

Scheduler Activations

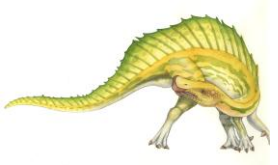
- هر دو مدل M:M و دو سطحی برای حفظ تعداد مناسب رشته های هسته اختصاص داده شده به برنامه نیاز به ارتباط دارند.
- به طور معمول از یک ساختار داده واسطه بین کاربر و رشته های هسته استفاده کنید - فرآیند سبک وزن.(LWP)
- به نظر می رسد یک پردازنده مجازی باشد که فرآیند می تواند رشته کاربر را برای اجرا برنامه ریزی کند.
- هر LWP به رشته هسته متصل است.
- چند LWP برای ایجاد؟
- فعال سازی های زمانبندی upcall ها را ارائه می دهند - یک مکانیزم ارتباط از هسته به هندلر upcall در کتابخانه رشته.
- این ارتباط به برنامه اجازه می دهد تا تعداد رشته های هسته صحیح را حفظ کند.





Operating System Examples

- Windows Threads
- Linux Threads

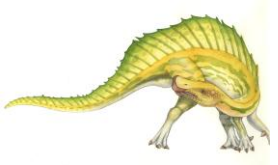


Windows Threads

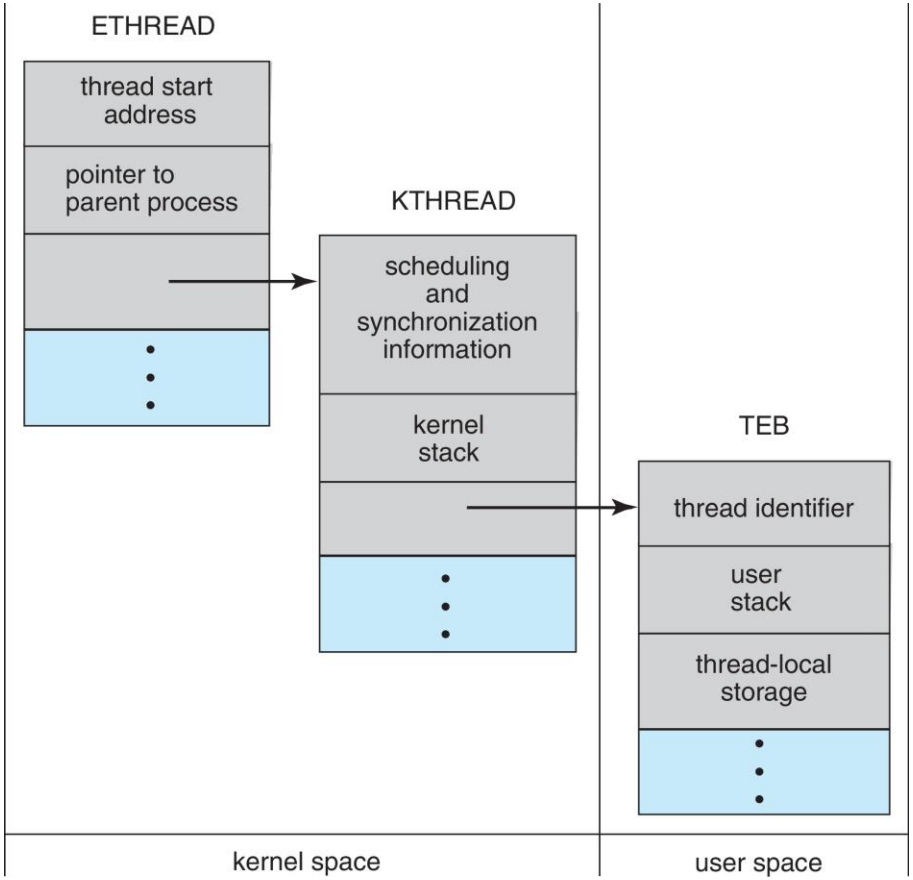
- ویندوز API - اصلی برای برنامه های کاربردی ویندوز.
- نقشه یک به یک، در سطح هسته را اجرا می کند.
- هر رشته حاوی:
  - یک شناسه رشته
  - مجموعه رجیستر نشان دهنده وضعیت پردازنده
  - پشته های جداگانه کاربر و هسته برای زمانی که رشته در حالت کاربر یا حالت هسته اجرا می شود.
  - ناحیه ذخیره سازی داده خصوصی که توسط کتابخانه های زمان اجرا و کتابخانه های پیوند پویا (DLL) استفاده می شود.
  - مجموعه رجیسترها، پشته‌ها و ناحیه ذخیره‌سازی خصوصی به عنوان محتوا (کانتکست) رشته شناخته می‌شوند.

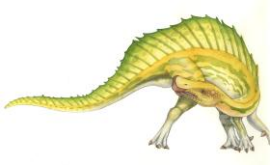
Windows Threads (Cont.)

- ساختارهای داده اصلی یک رشته شامل موارد زیر است:
- **ETHREAD (بلاک رشته اجرایی** - (شامل اشاره‌گر به فرآیندی که رشته به آن تعلق دارد و همچنین اشاره‌گر به KTHREAD در فضای هسته.
- **KTHREAD (بلاک رشته هسته** - (اطلاعات زمان‌بندی و همگام‌سازی، پشته حالت هسته، اشاره‌گر به TEB در فضای هسته.
- **TEB (بلاک محیط رشته** - (شناسه رشته، پشته حالت کاربر، ذخیره‌سازی محلی رشته در فضای کاربر.



Windows Threads Data Structures





Linux Threads

- لینوکس به جای رشته از واژه **وظیفه** استفاده می‌کند.
- ایجاد رشته از طریق فراخوان سیستمی clone() انجام می‌شود.
- clone() به یک وظیفه فرزند اجازه می‌دهد تا فضای آدرس وظیفه والد (فرآیند) را به اشتراک بگذارد.
- پرچم‌ها رفتار را کنترل می‌کنند.

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.

- ساختار task_struct به ساختارهای داده فرآیند (مشترک یا منحصر به فرد) اشاره می‌کند.

پایان فصل ۴