



操作系统

Operating system

胡燕

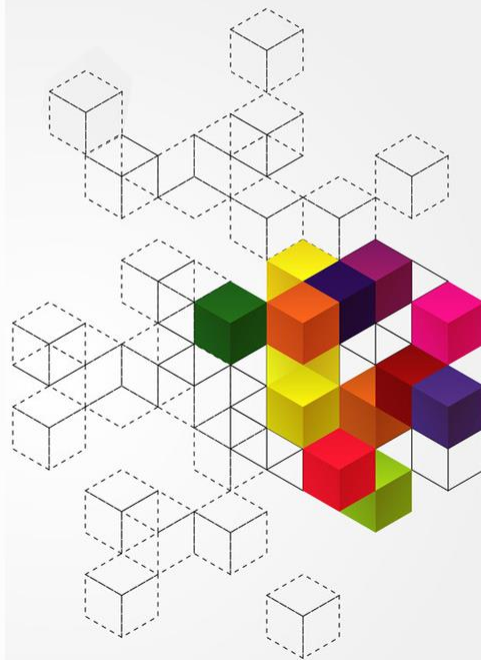
大连理工大学

一、什么是线程

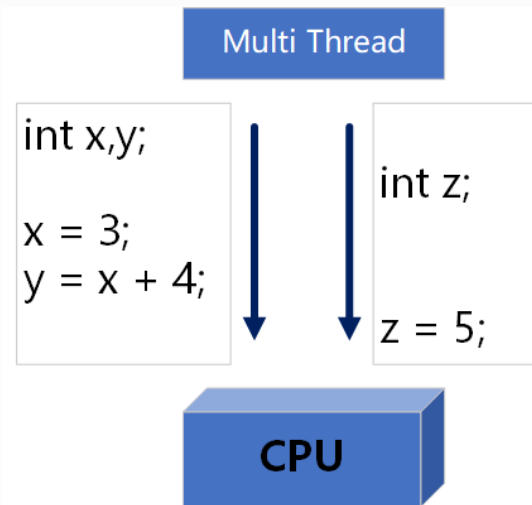
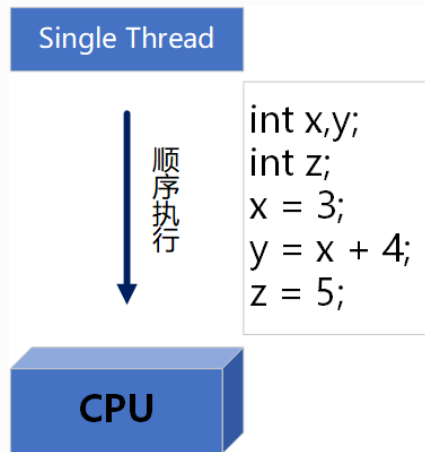
二、线程结构

三、线程 v.s. 进程

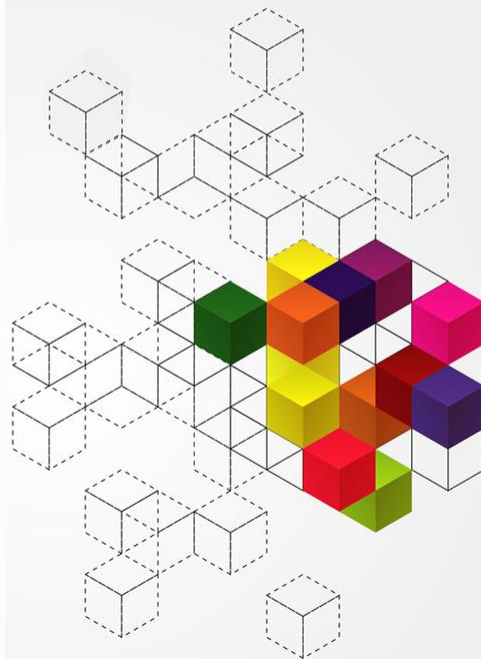
四、线程应用场景



一、什么是线程？

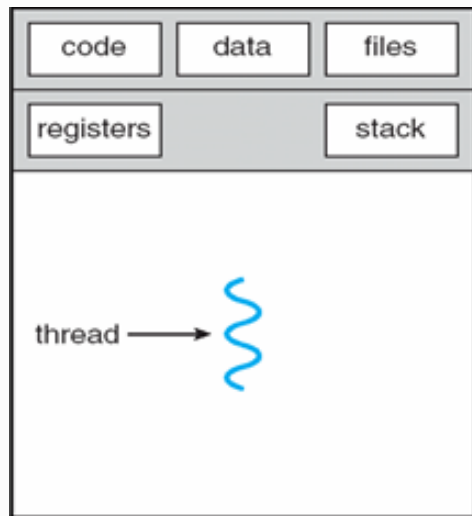


- **线程**是将进程的计算任务进一步细分后得到的更细粒度的计算单位

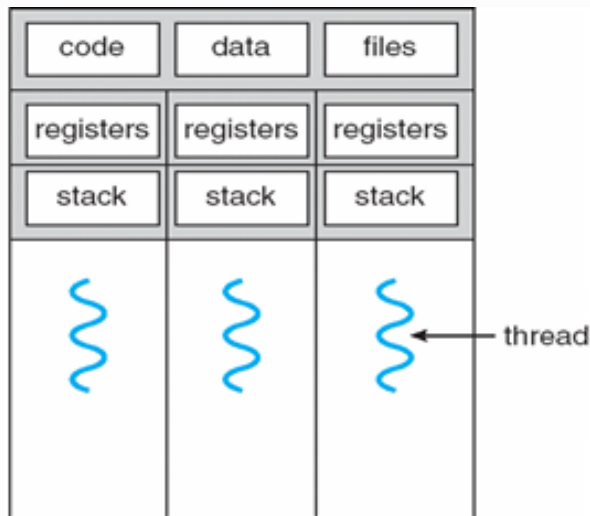


一、什么是线程？

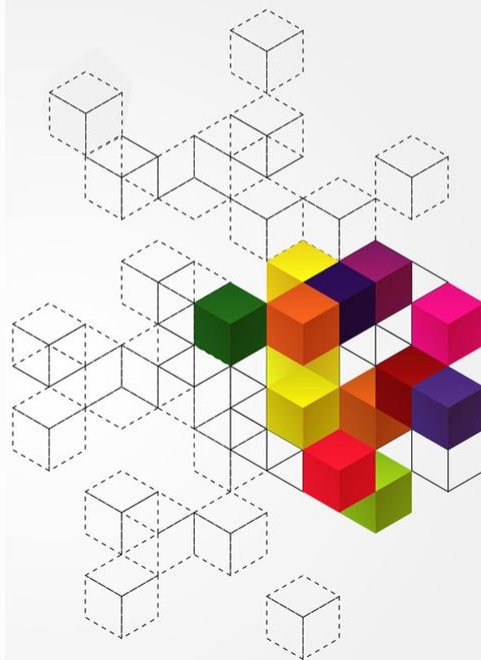
线程概念示意图：



单线程进程

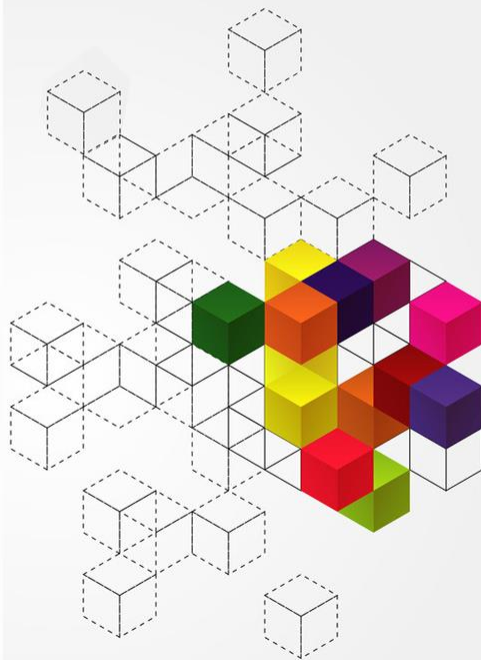


多线程进程



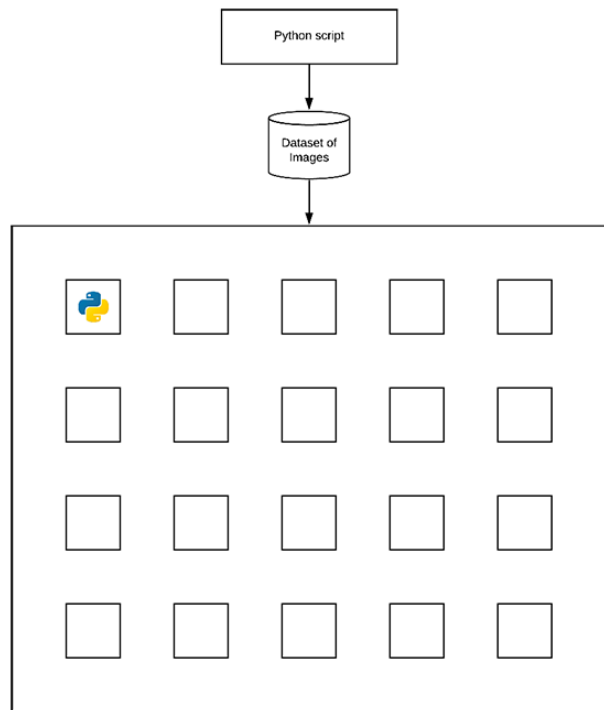
一、什么是线程？

- **线程** (Thread)
 - 台湾同胞称为: 多线、多执行绪(千头万绪之意)
 - **多线程**: 进程中的代码流, 从1个被拆分为至少2个

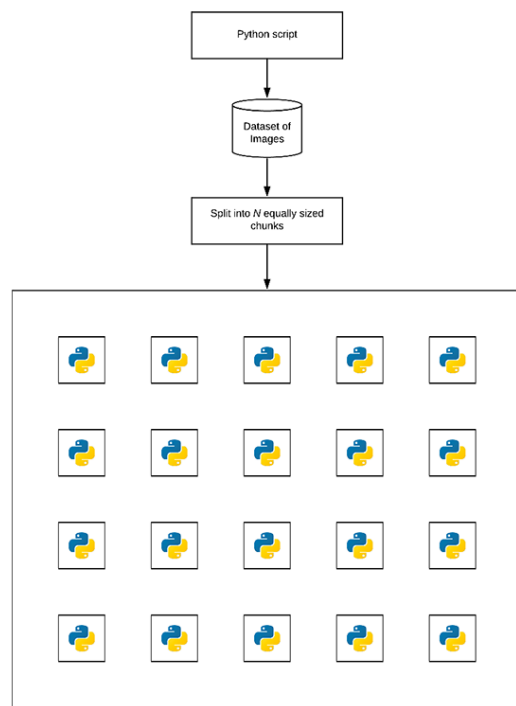


一、什么是线程？

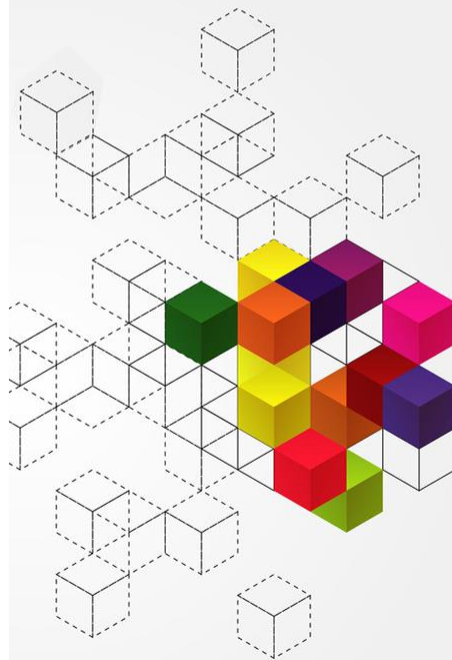
- **线程** (Thread): 可以带来的好处: 示例



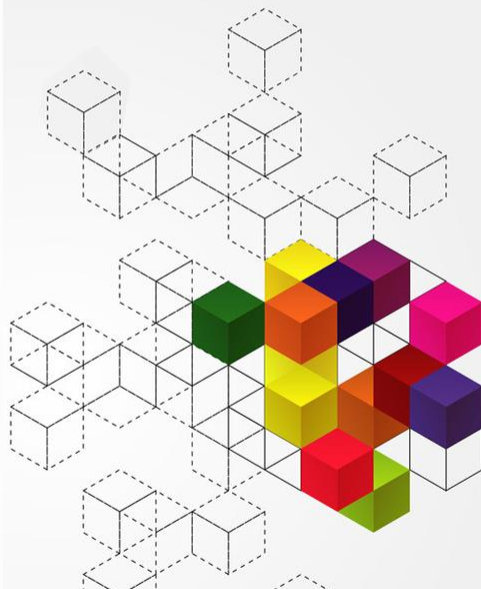
3 GHz Intel Xeon W processor w/ 20 cores



3 GHz Intel Xeon W processor w/ 20 cores



在CPU单核条件下，引入线程增加并发粒度，有无好处？为什么？

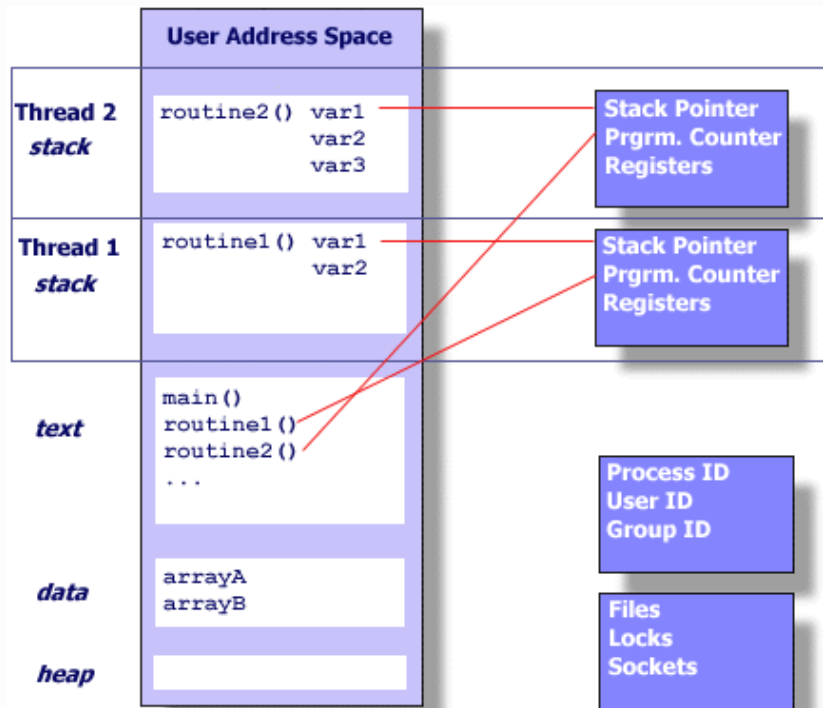


正常使用主观题需2.0以上版本雨课堂

作答

一、什么是线程？

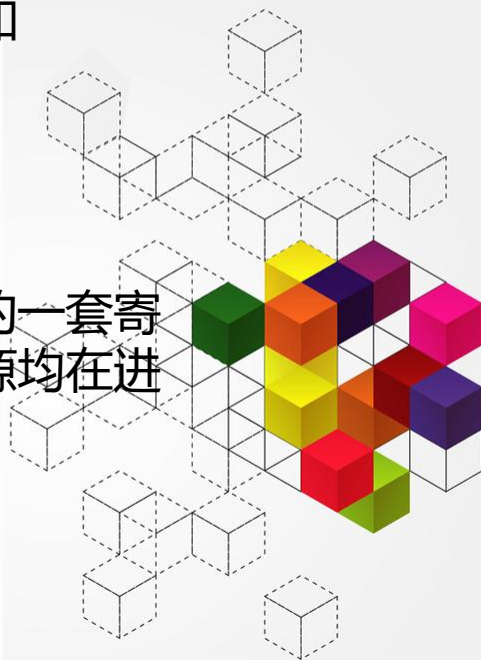
多线程的进程地址空间示意



线程的通常代码模式:

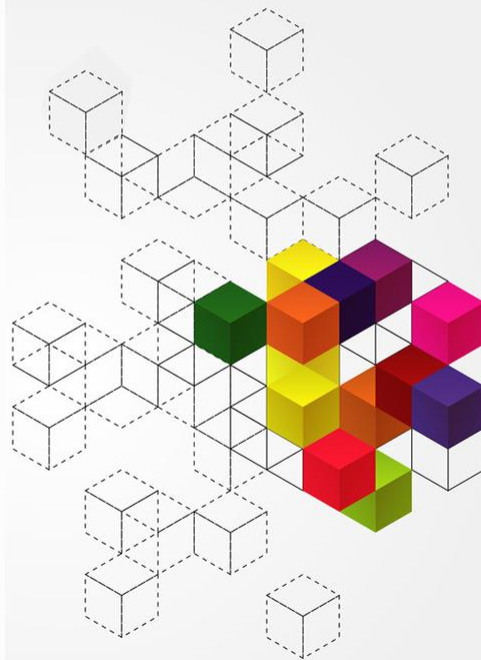
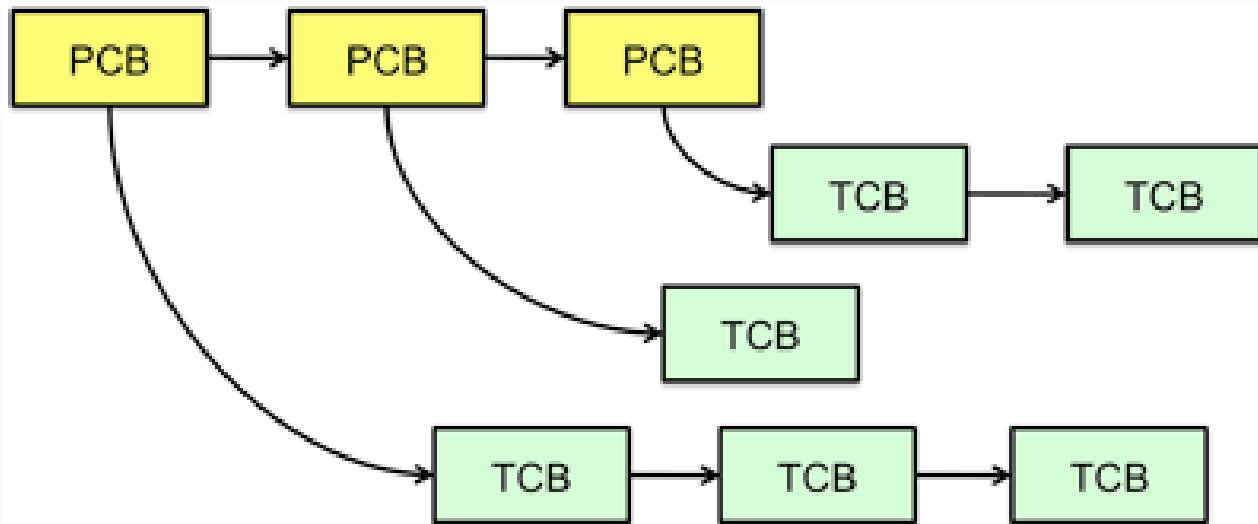
- (1)主线程main
- (2)其他的线程过程,如 `routine1`, `routine2`

所有的线程有自己的一套寄存器 and 栈, 其他资源均在进程地址空间共享

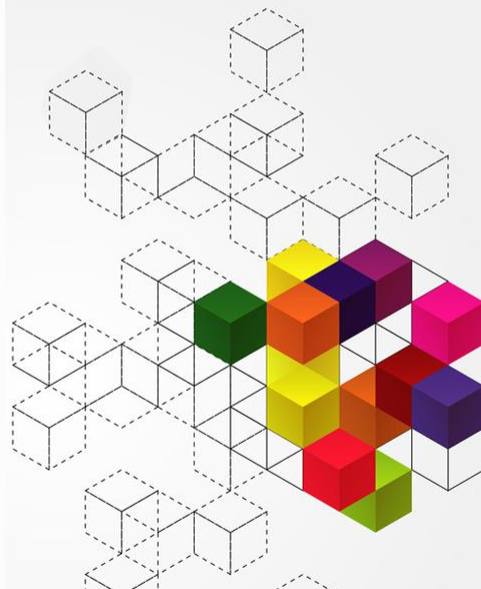


二、线程结构

- OS内核中的线程管理数据结构：TCB
 - TCB(Thread Control Block)



请尝试设计线程控制块的数据结构。



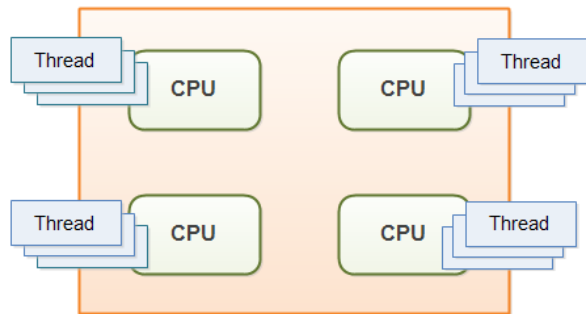
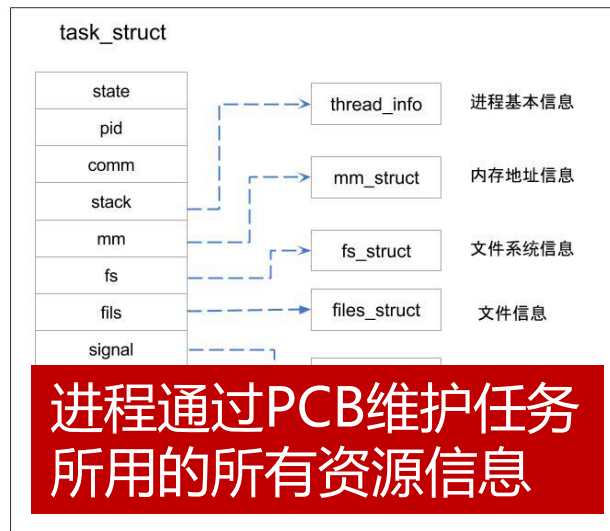
正常使用主观题需2.0以上版本雨课堂

作答

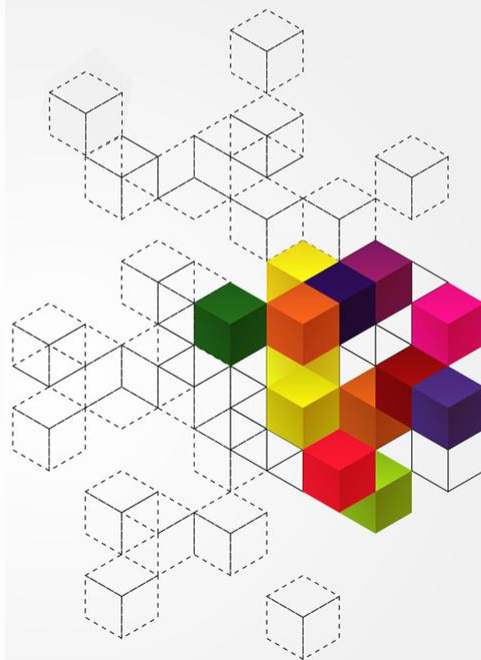
三、线程 v.s. 进程

• 线程与进程之间的联系

- 进程：拥有资源
- 线程：使用所隶属进程的资源进行计算



线程利用CPU资源
完成计算任务



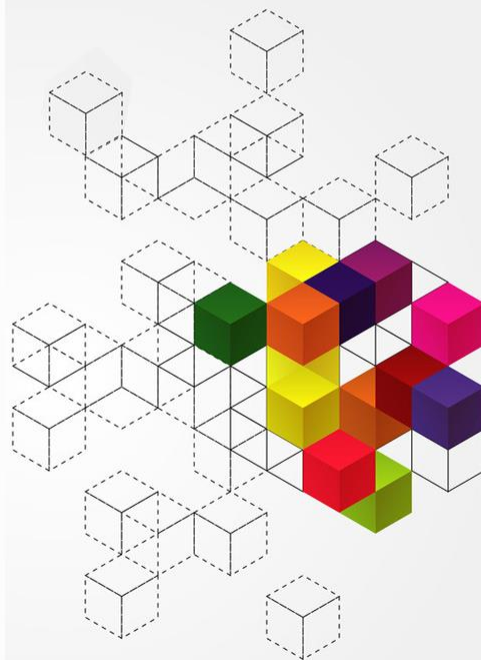
MultiTasking v.s. MultiThreading

三、线程 v.s. 进程

- 线程与进程之间的联系

- 进程：拥有资源
- 线程：使用所隶属进程的资源进行计算

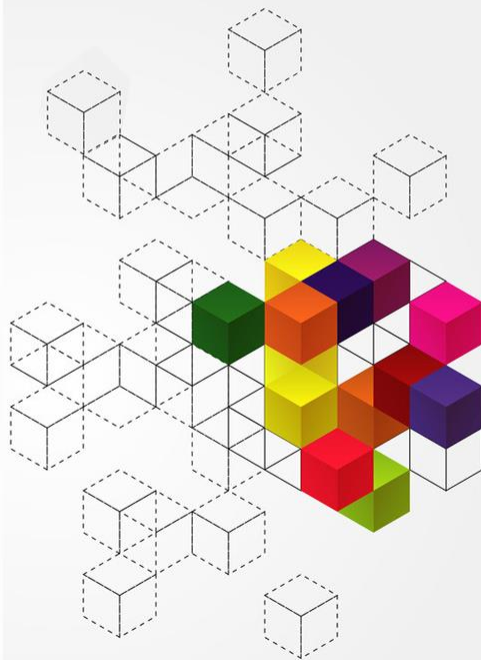
进程是舞台，线程是演员
进程是平台，线程是人才



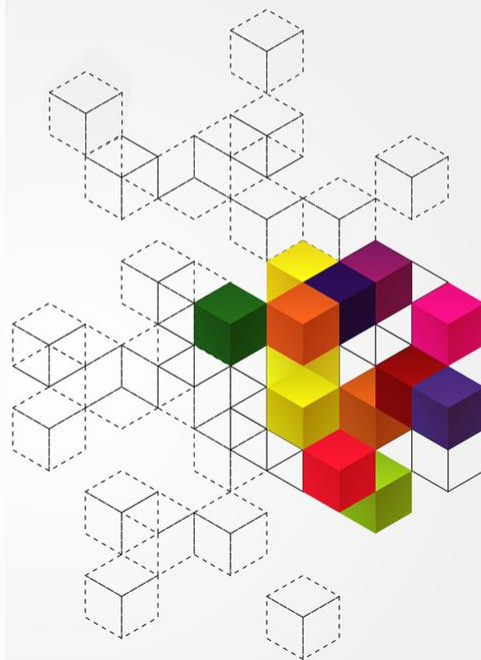
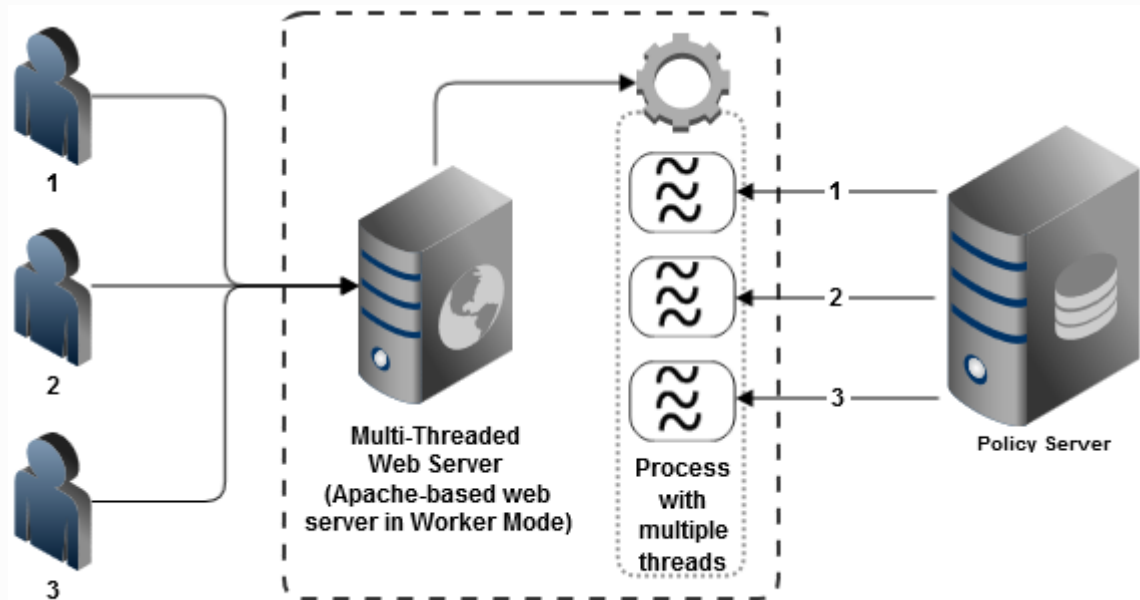
三、线程 v.s. 进程

- **线程的优势**

- 相比于进程，线程之间的切换代价要小得多

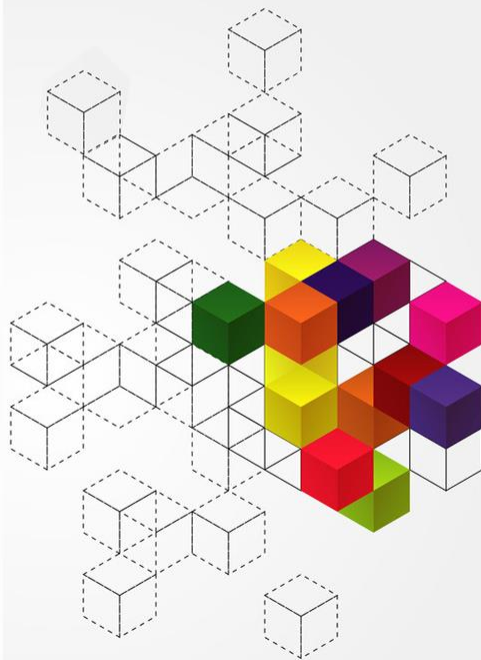


四、线程应用场景



本讲小结

- 什么是线程
- 线程结构
- 线程v.s.进程
- 线程典型应用场景

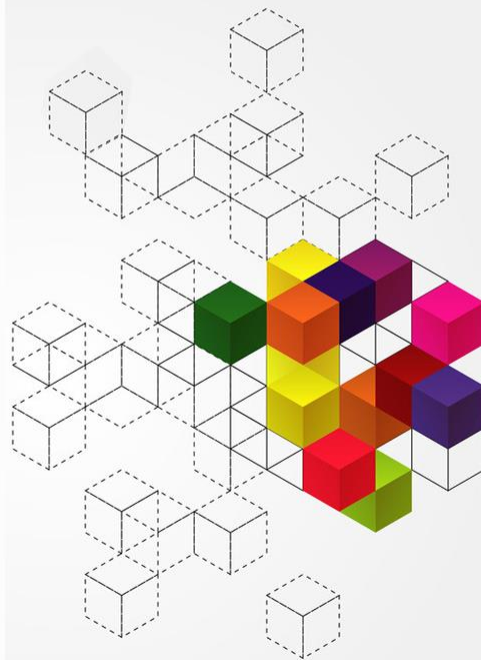


一、线程分类

二、几种典型的线程模型


三、Linux线程模型

四、Solaris线程模型




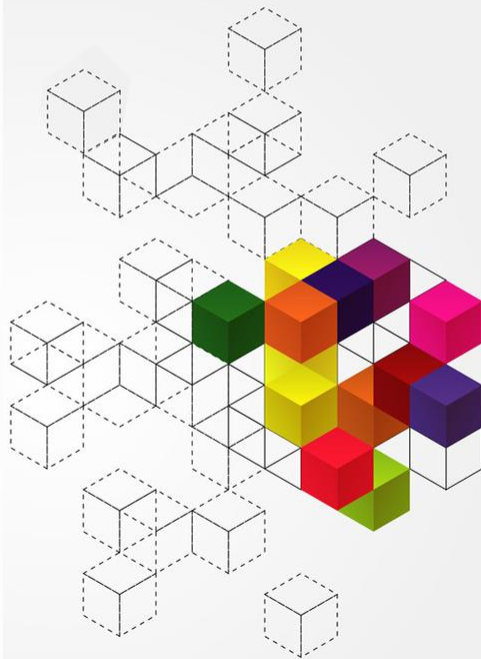
一、线程分类

用户级线程

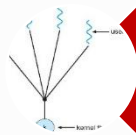
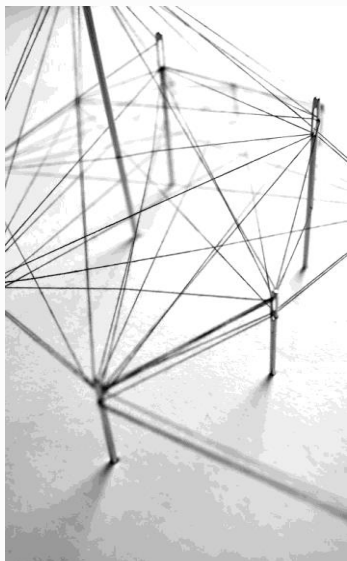
- 
- 在用户态以线程库的形式实现
 - 对用户级线程的操作通过调用用户态线程库API进行

内核级线程

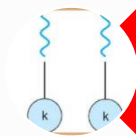
- 
- 在内核态实现，OS内核直接管理
 - 线程的创建由系统调用完成



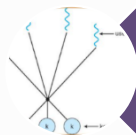
二、几种典型的线程模型



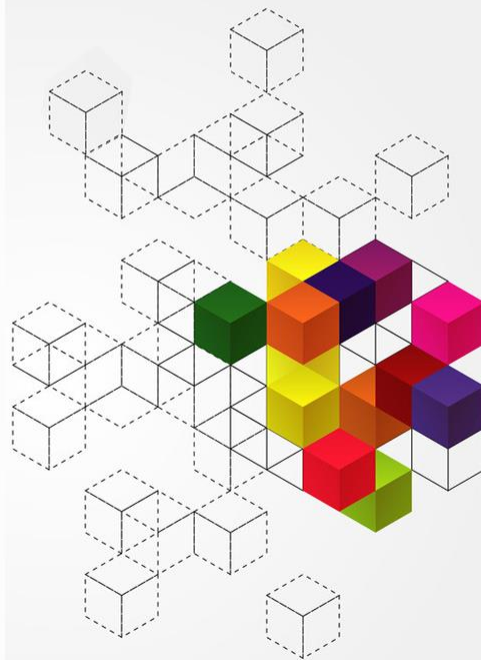
M:1



1:1

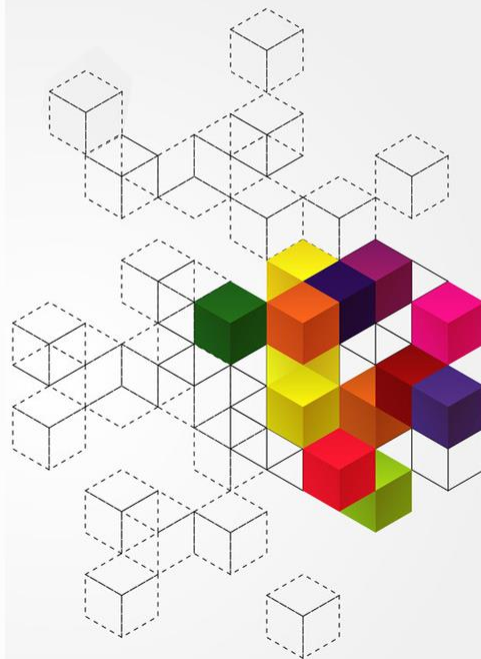
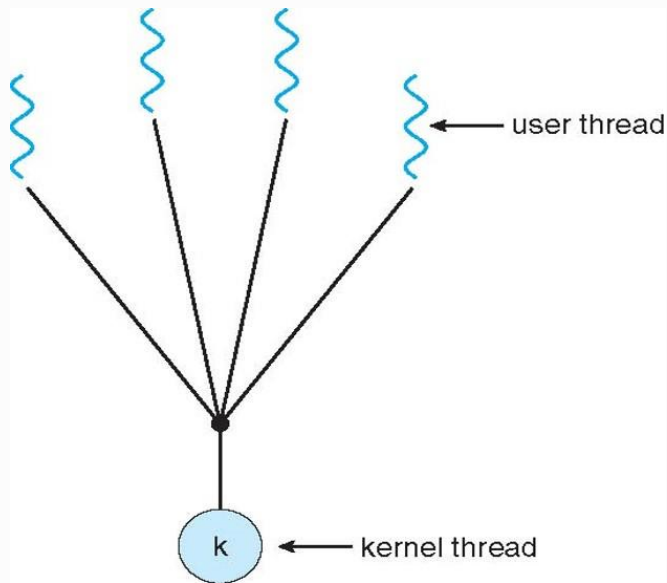


M:N



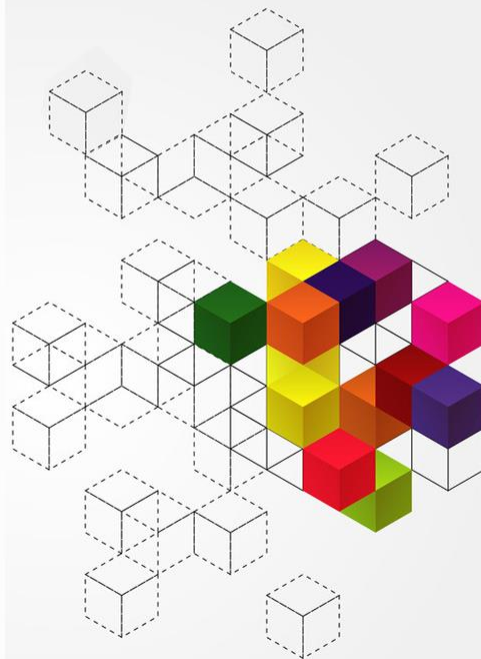
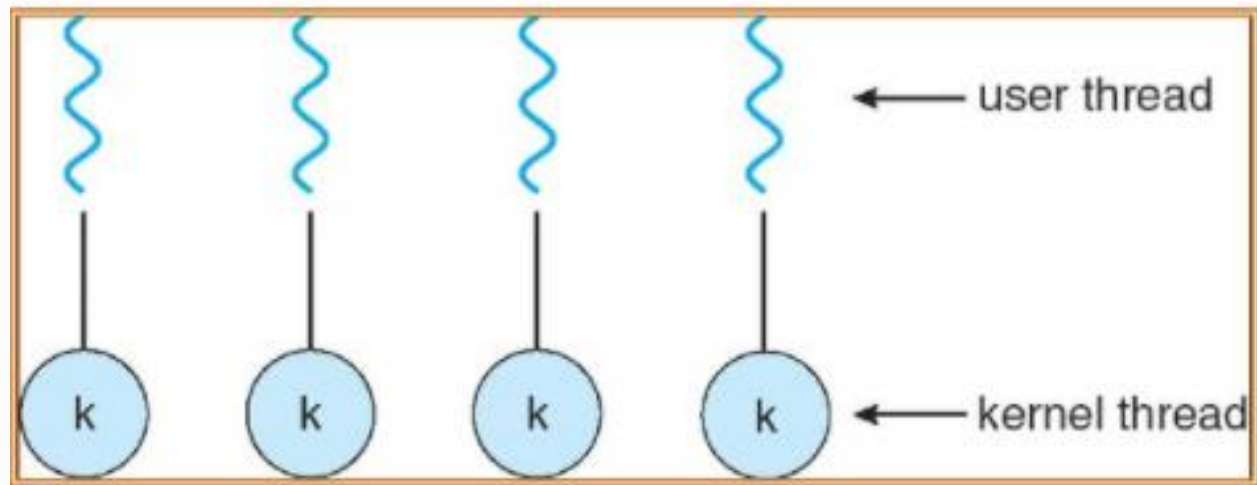
二、几种典型的线程模型 -M:1模型

- 多个用户级线程绑定到一个内核级线程(M:1)



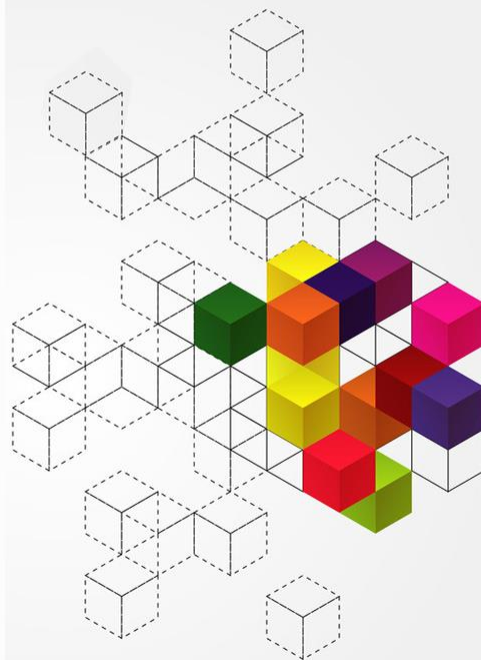
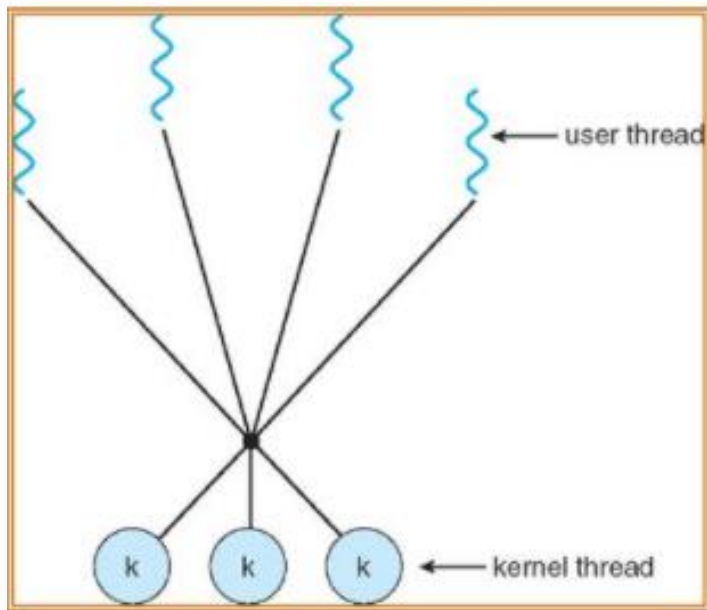
二、几种典型的线程模型 -1:1模型

- 将1个用户级线程绑定到1个内核级线程(1:1)



二、几种典型的线程模型 -M:N模型

- 将多个用户级线程绑定到多个内核级线程(M:N)



三、Linux线程模型

LinuxThreads

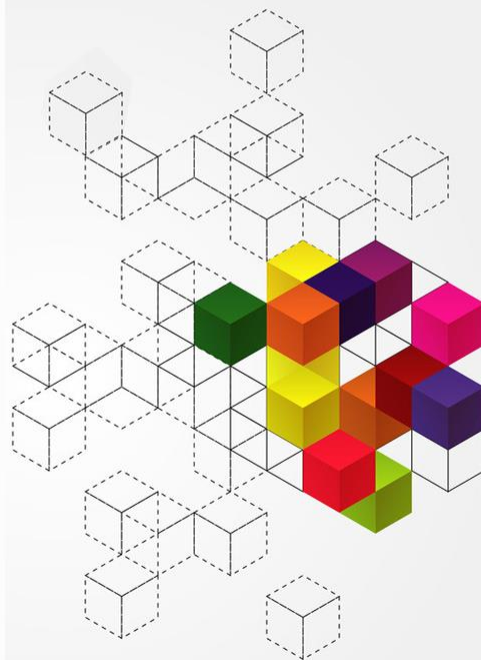
最早的模型，部分
实现Posix
Threads

NGPT

Next Generation
Posix Threads(已
终止)

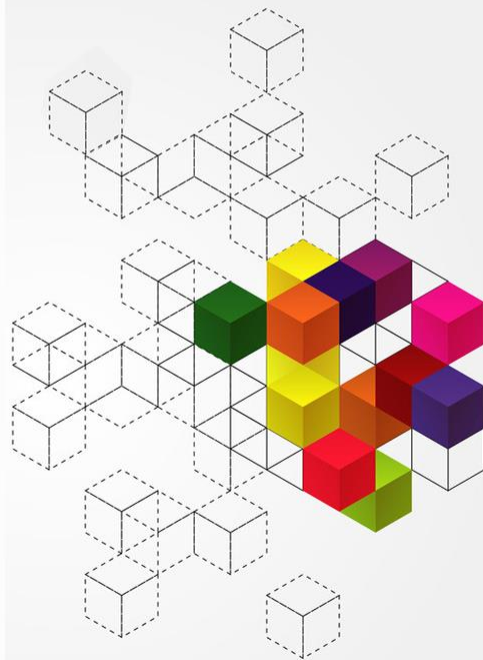
NPTL

Native Posix
Thread Library,
自2.6内核以来使
用的线程模型

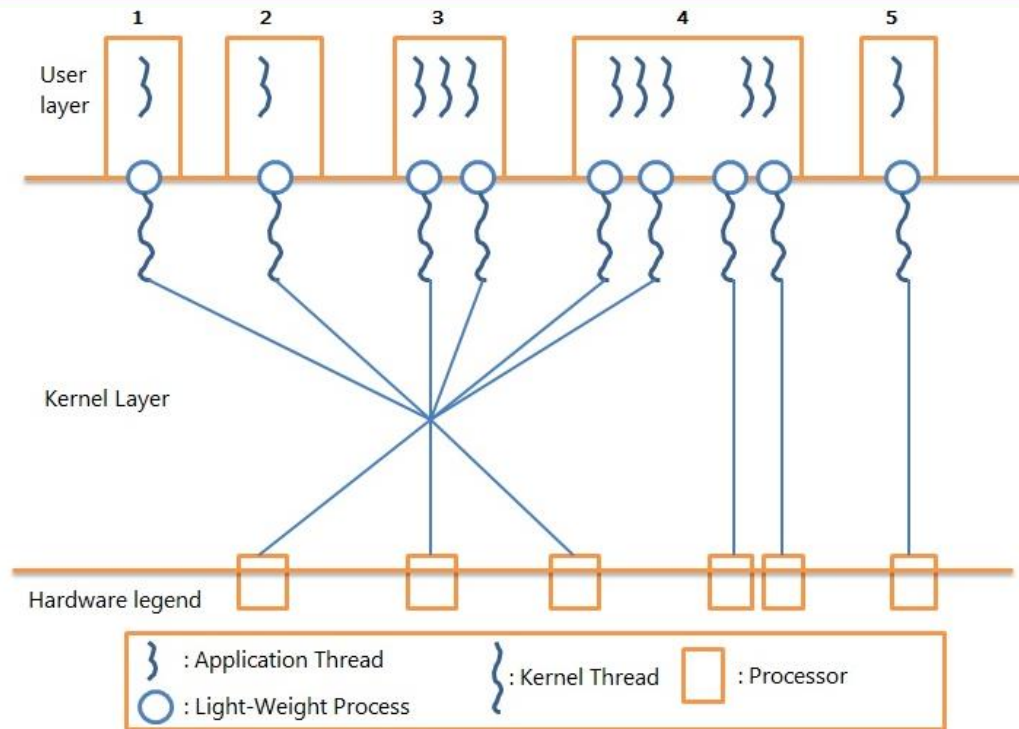


三、Linux线程模型

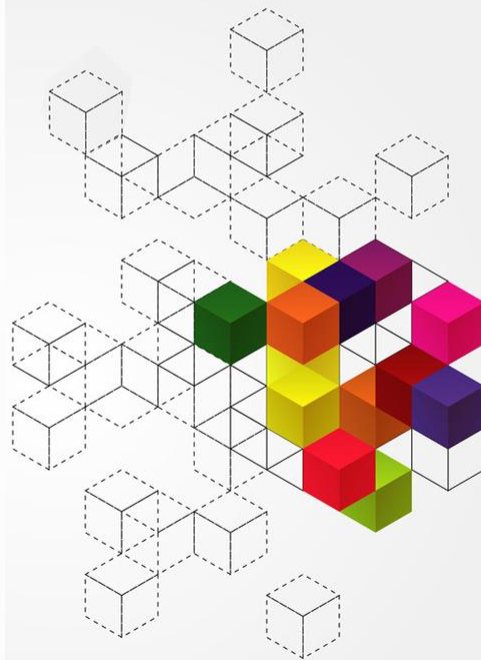
- 在NPTL中，采用的是1：1线程模型
 - 当调用pthread库创建一个线程时，pthread_create会实际调用clone系统调用，最终会创建一个LWP（轻量级进程）和一个内核线程
 - 内核线程具有对应的task_struct结构，是个独立的内核调度单元



四、Solaris线程模型

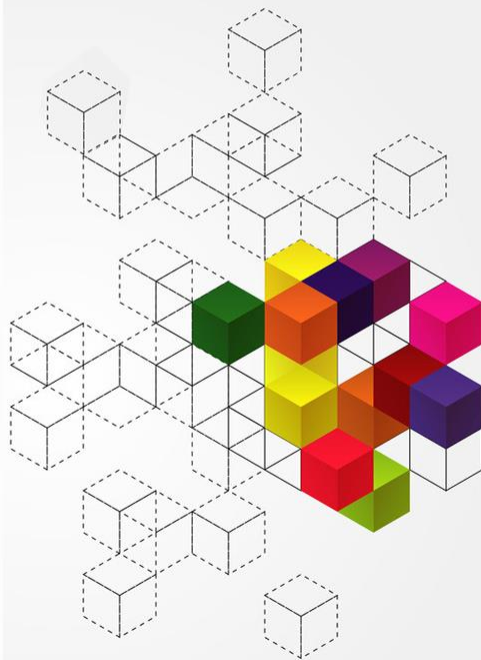


- Solaris 2-8: M:N混合模型
- Solaris9-10: 默认线程模型改为1:1模型



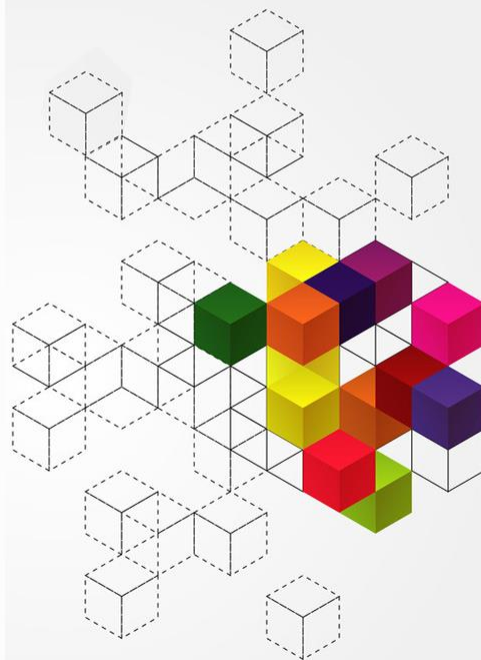
本讲小结

- 线程分类
- 几种典型的线程模型
- Linux线程模型
- Solaris线程模型



一、Windows线程编程API

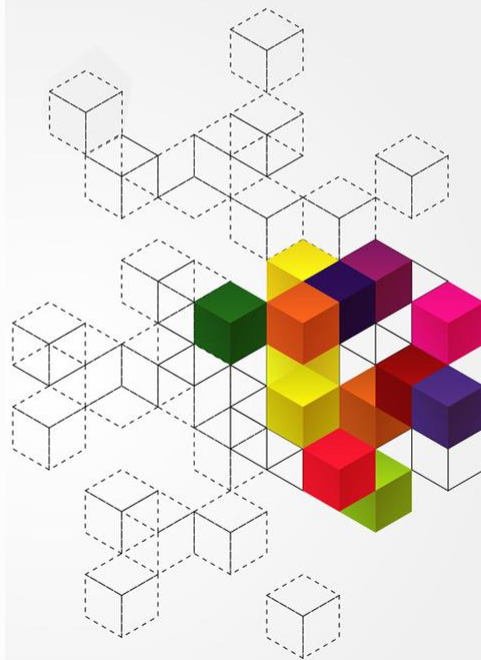
二、Windows线程编程示例



一、Windows线程编程API

- 进程中的任意线程中都可以创建新线程
- 主线程在进程加载时自动创建
- 每个线程有自己的入口点函数
- 主线程的进入点函数

进入点	应用程序类型
WinMain	要求ANSI字符和字符串的GUI应用程序
wWinMain	要求Unicode字符和字符串的GUI应用程序
Main	要求ANSI字符或字符串的CLI应用程序
Wmain	要求Unicode字符和字符串的CLI应用程序

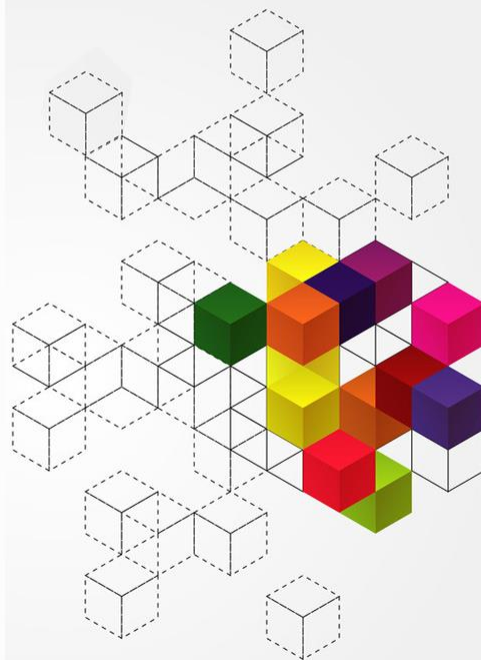


一、Windows线程编程API

- Windows线程入口点代码样式

```
DWORD WINAPI ThreadFunc(PVOID pvParam) {  
    DWORD dwResult = 0;  
    ...;  
    ...;  
    ...;  
    return dwResult;  
}
```

- 线程函数的返回值是线程的EXIT CODE
- 线程函数应尽可能使用函数传递的参数和局部变量

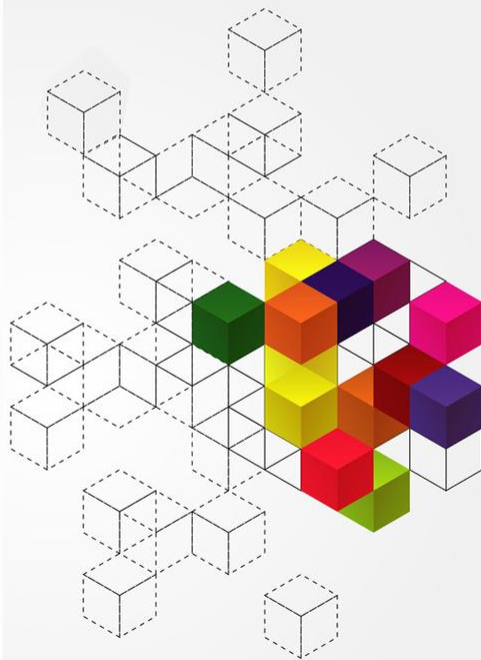


一、Windows线程编程API

- CreateThread: 创建线程的API函数

```
HANDLE CreateThread {  
    PSECURITY_ATTRIBUTES psa,  
    ...;  
    ...;  
    ...;  
    return dwResult;  
}
```

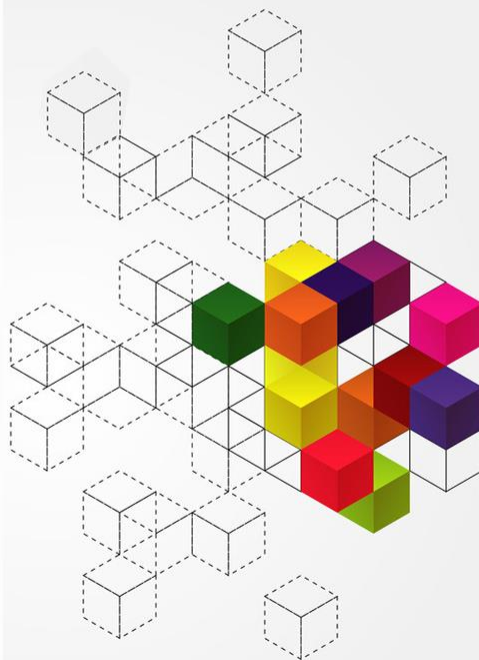
- 当调用CreateThread创建线程时，系统创建一个线程内核对象，该内核对象是用来管理进程的内核数据结构
- 在进程地址空间分配内存，供线程的堆栈使用



二、Windows线程编程示例

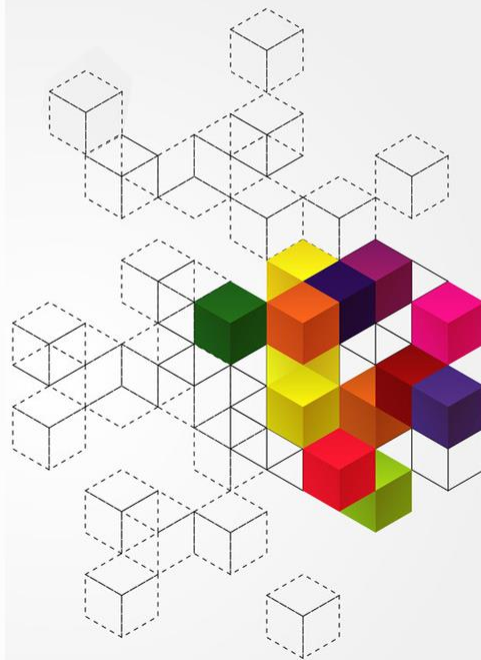
示例代码

```
#include <windows.h>
#include <iostream>
using namespace std;
DWORD WINAPI ThreadFunc(PVOID pvParam){
    cout << "create thread says 'Hello World!'" << endl;
    return 0;
}
int main(){
    HANDLE h = CreateThread(NULL,0,ThreadFunc,NULL,0,NULL);
    Sleep(100);
    cout << "Main thread" << endl;
    getchar();
    return 0;
}
```



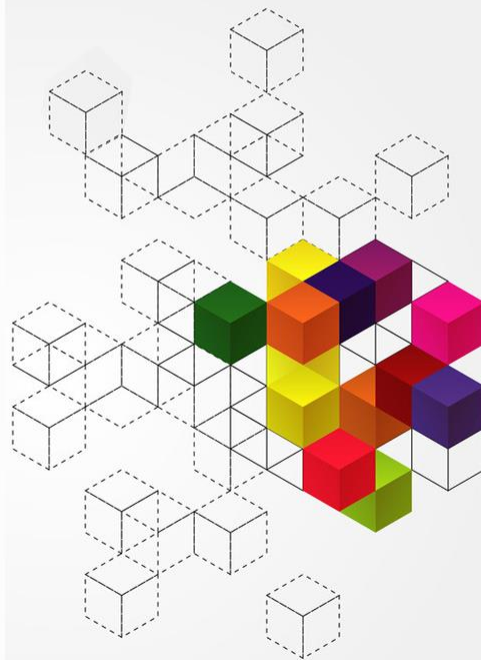
本讲小结

- Windows线程编程API
- Windows线程编程示例



一、Linux线程编程API

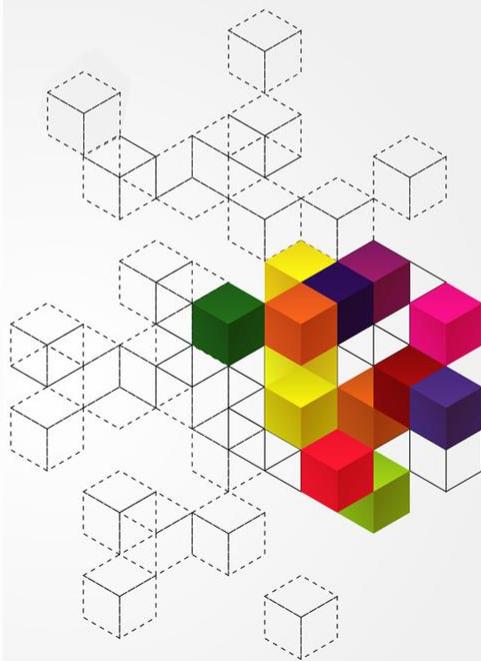
二、Linux线程编程示例



一、Linux线程编程API

- Linux Pthread库

- Pthread库是符合IEEE 1003.1c的POSIX标准规范的线程库
- Pthread线程库中有60多个函数，包括线程创建、线程终止、线程同步等操作
- Pthread库在Windows也有移植实现

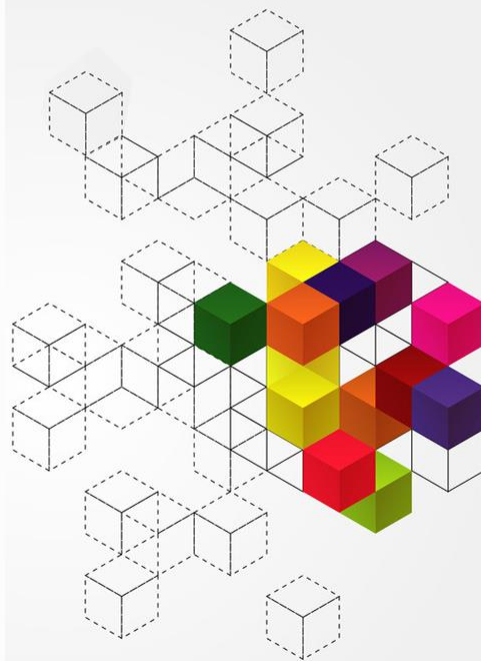


一、Linux线程编程API

- **pthread_create**: 创建线程的API函数

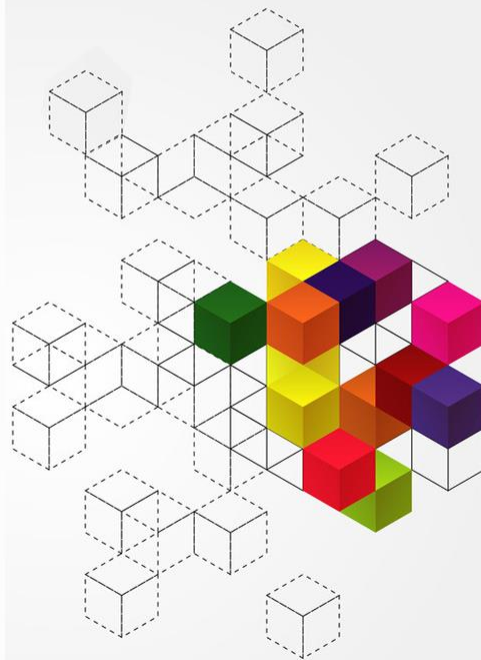
```
int pthread_create(  
    pthread_t *restrict tidp,  
    const pthread_attr_t *restric attr,  
    void *(*start_rtn)(void *),  
    void *restric arg  
);
```

- 返回值：若成功返回0，否则返回出错编号
- 返回成功时，由tidp指向的内存单元被设置为新创建线程的线程ID



二、Linux线程编程示例

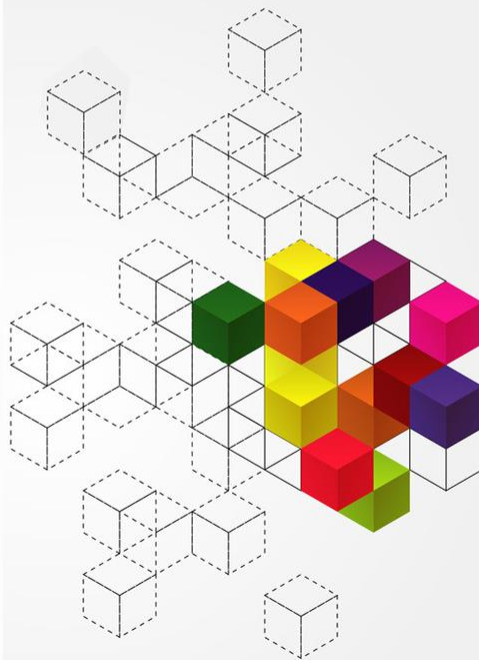
```
#include <pthread.h>
#include <stdio.h>
void *runner(void *param);
int sum;
main( ) {
    pthread_t tid; /* the thread ID */
    pthread_attr_t attr; /* set of thread attrs */
    pthread_attr_init(&attr); /* get the default attr */
    pthread_create(&tid, &attr, runner, "10"); /* create the thread */
    pthread_join(tid, NULL); /* wait for the thread to exit */
    printf("sum = %d\n", sum);
}
void *runner(void *param) {
    int upper = atoi(param);
    sum = 0;
    for (i=0; i<upper; i++) sum += i;
    pthread_exit(0);
}
```



二、Linux线程编程示例

- 线程创建示例
 - 线程函数部分

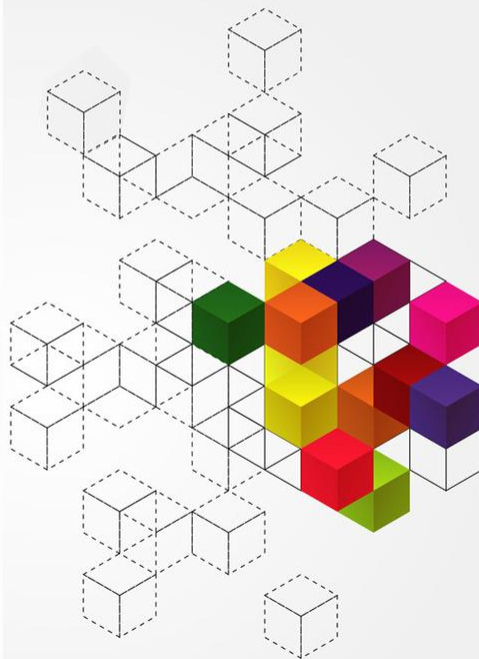
```
1 /* thread_create.c */
2 #include<stdio.h>
3 #include<stdlib.h>
4 #include<pthread.h>
5
6 /*线程函数1*/
7 void *mythread1(void)
8 {
9     int i;
10    for(i=0;i<5;i++)
11    {
12        printf("I am the 1st pthread,created by mybeilef321\n");
13        sleep(2);
14    }
15 }
16 /*线程函数2*/
17 void *mythread2(void)
18 {
19     int i;
20    for(i=0;i<5;i++)
21    {
22        printf("I am the 2st pthread,created by mybelief321\n");
23        sleep(2);
24    }
25 }
26
```



二、Linux线程编程示例

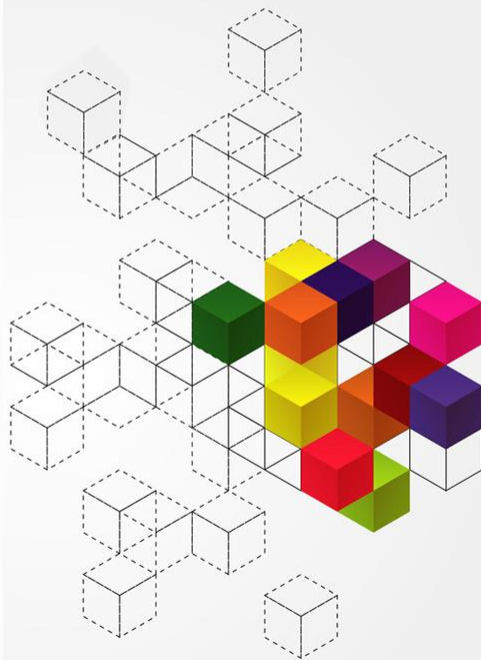
- 线程创建示例
 - 主函数部分

```
27 int main()
28 {
29     pthread_t id1,id2; /*线程ID*/
30     int res;
31     /*创建一个线程，并使得该线程执行mythread1函数*/
32     res=pthread_create(&id1,NULL,(void *)mythread1,NULL);
33     if(res)
34     {
35         printf("Create pthread error!\n");
36         return 1;
37     }
38     /*创建一个线程，并使得该线程执行mythread2函数*/
39     res=pthread_create(&id2,NULL,(void *)mythread2,NULL);
40     if(res)
41     {
42         printf("Create pthread error!\n");
43         return 1;
44     }
45     /*等待两个线程均推出后，main()函数再退出*/
46     pthread_join(id1,NULL);
47     pthread_join(id2,NULL);
48
49     return 1;
50 }
```



二、Linux线程编程示例

- 线程创建示例
 - 编译: `gcc thread_create.c -o thread_create -lpthread`
 - 执行: `./thread_create`





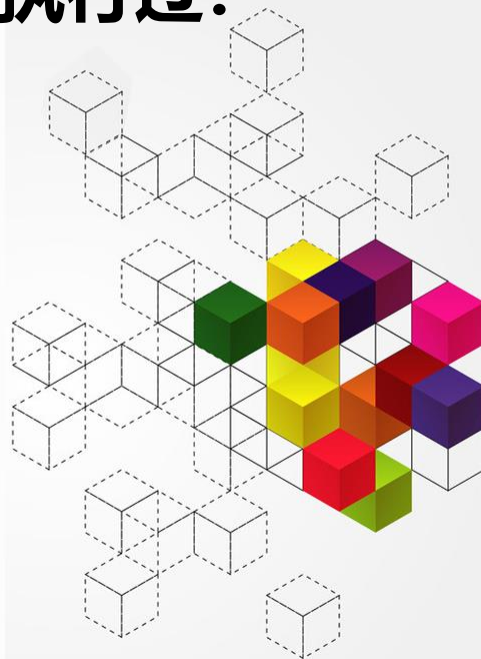
基于pthread的示例代码有没有实际编译执行过？

A

有

B

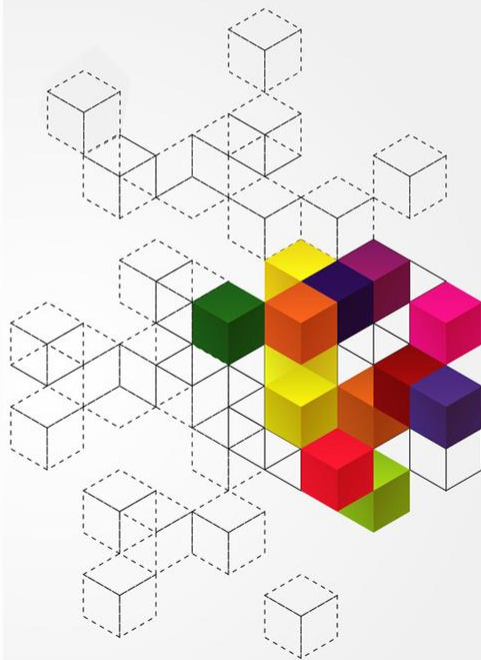
没有



提交

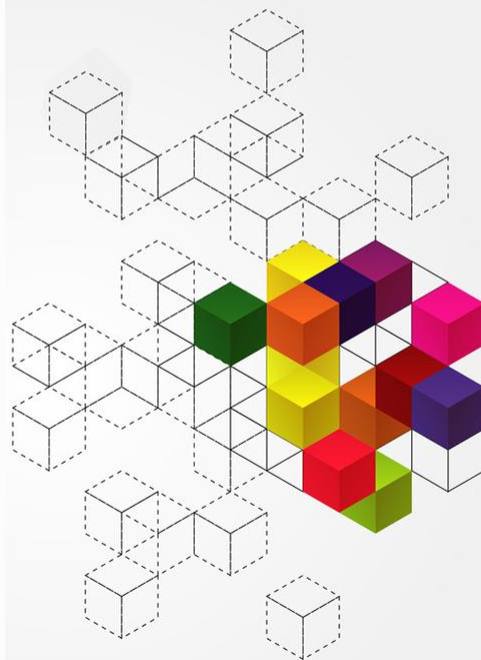
二、Linux线程编程示例

- **线程练习1:** 编写程序, 用pthread create函数创建一个线程, 并在新线程中打印此线程的id和其所属进程的id



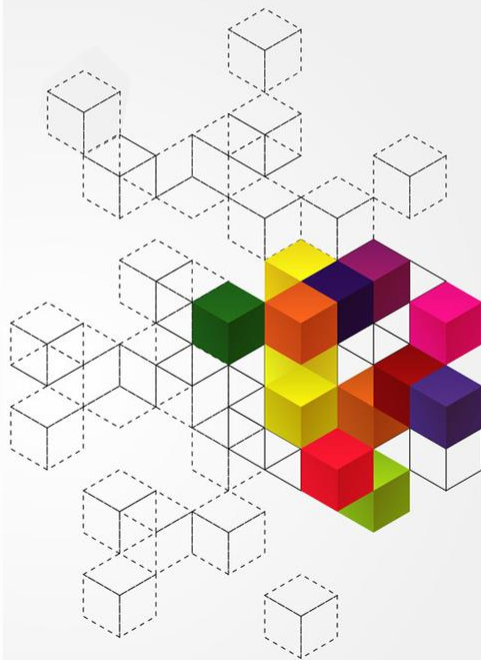
二、Linux线程编程示例

- **线程练习2:** 编写程序，创建2个线程，每个线程中各有一个循环，分别输出This is thread 1和 This is Thread 2各10次。



二、Linux线程编程示例

- **线程练习3：**编写程序，创建一个线程，从1到n累加后，将结果打印输出。n是变量，由主线程传入。

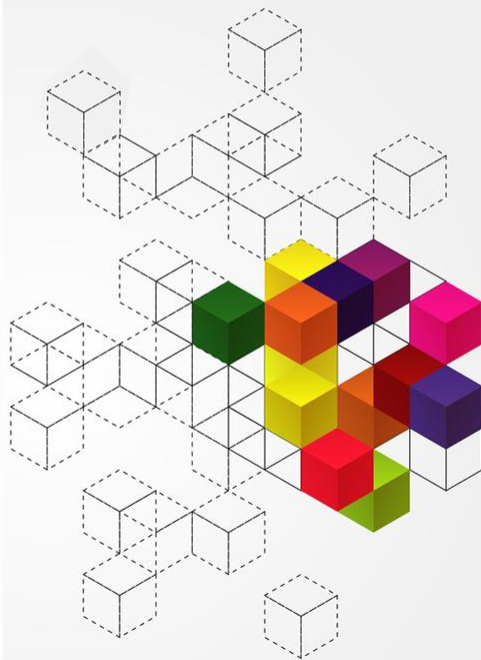


二、Linux线程编程示例

- **线程练习4:** 编写程序, 主线程中创建student数组

```
struct student {  
    char name[16];  
    char id[16];  
    float score;  
}
```

- 用一个线程来计算平均分。

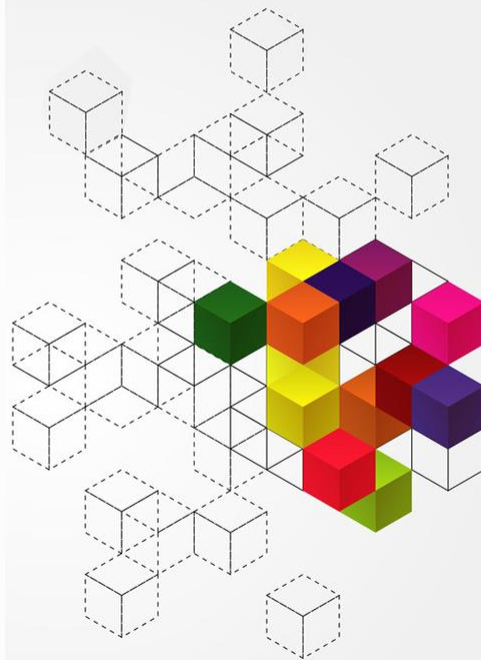


二、Linux线程编程示例

- **课后练习题：**编写程序，主线程中创建student数组，其类型为

```
struct student {  
    char name[16];  
    char id[16];  
    float score;  
}
```

- 设数组大小为200，用10个线程来协助计算，并最终由主线程来统计平均分。



本讲小结

- Linux线程编程API
- Linux线程编程示例

