第四章 线程

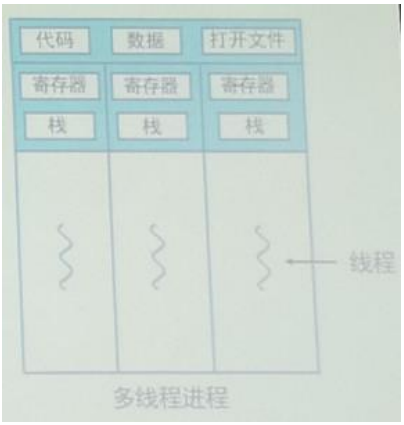
4.1 概述

1、线程的基本概念

①线程是进程的一部分，描述指令执行流状态，是进程中的指令执行流的最小单元，是 CPU 调度的基本单位。

②进程：资源分配角色；　　线程：CPU 调度角色

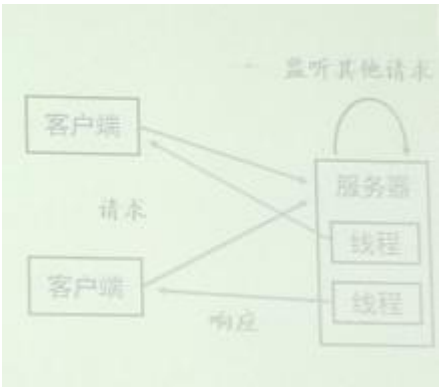
2、线程需要的资源



共享：代码、数据、打开文件、**全局变量、堆**

独有：寄存器组、栈

3、多线程的服务器框架



传统进程方式---多进程方式---多线程方式

大多是操作系统**内核都是多线程**的

4、线程优点和缺点

①响应性：即使部分线程阻塞，其他线程仍可继续执行

②资源共享：线程共享所属进程的内存和文件；进程通过进程通信机制和内存共享技术共享。

③经济性：线程创建和切换开销较小
④可扩展性：线程可在多处理核上并行运行
缺点：一个线程崩溃导致所属进程的所有线程崩溃

4.2 多核编程

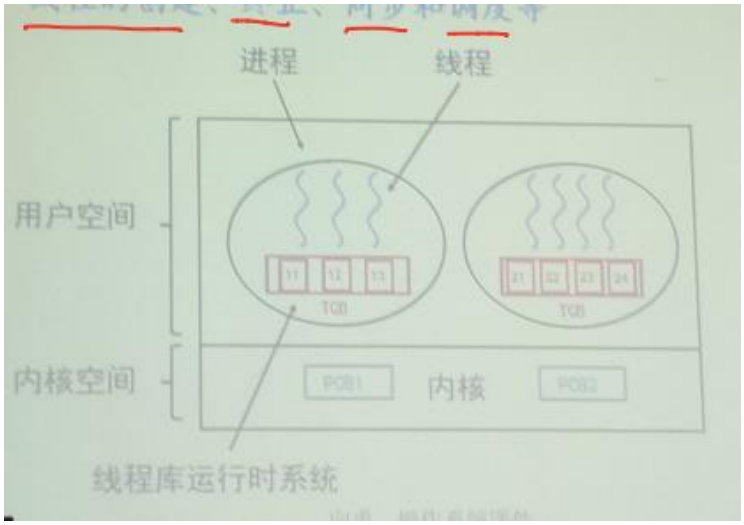
1、并行和并发
并行性：多个事件在同一时刻发生
并发性：多个事件在同一时间段内发生
CPU 包含多个内核

2、编程挑战
①识别任务
②平衡
③数据分割
④数据依赖
⑤测试与调试

3、并行类型
①数据并行
将数据分布到计算核上，每个核执行相同的操作
②任务并行
将任务分布到计算核上，每个核执行不同的操作；每个线程可以操作相同的、不同的数据。
③混合型
绝大多数情况

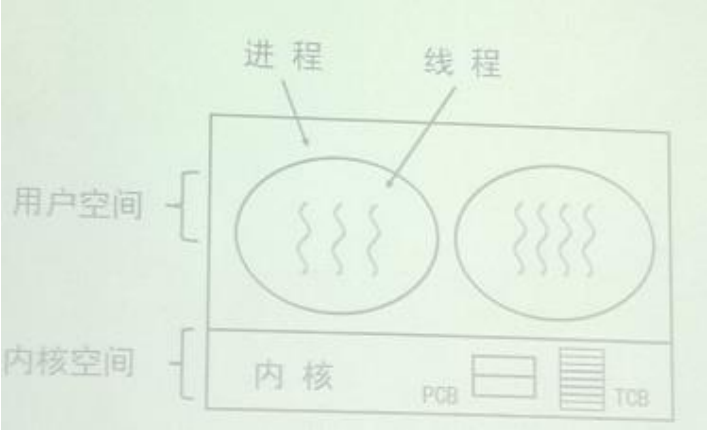
4.3 多线程模型

1、用户线程



①特点：解决了上下文切换的开销
②TCB：在进程的用户空间
③管理线程：用户级的线程库函数
④优点：无需用户态/内核态相互转换，速度快
⑤缺点：内核以进程为单位调度，无法发挥多核优势。
（具体体现在：一个线程阻塞，整个进程等待；一个线程运行，进程中的其他线程无法运行；时间片分配给进程，多线程则每个线程就慢）

2、内核线程



①特点：发挥多处理器的并发优势
②TCB：在内核空间
③管理线程：OS 内核
④优点：内核以线程为单位调度，同一进程内的多个线程的并行执行
⑤缺点：线程的调度和同步通过系统调用来实现，开销大

3、多线程模型
①多对一（没人用了）
②一对一（使用最多）
③多对多（折中方案，性能最佳，但是实现复杂）

4.4 线程库（实验必考）

1、定义
为程序元提供创建和管理线程的 API
2、用户级线程库和内核级线程库
用户级线程库：通过用户空间内的函数来调用
内核级线程库：系统调用

2、三种主要线程库
①POSIX Pthreads（用户级和内核级的线程库）
②Windows（内核级的线程库）
③Java(不考)
①和②中的线程共享全局变量

3、多线程创建的策略

①异步线程

父子线程独立并发运行，导致线程之间很少有数据共享

②同步线程

父进程等待所有子进程终止，才能恢复执行；导致线程之间存在大量的数据共享

4、POSIX Pthreads

①大多数 Unix 系统实现该规范

②

```
#include <pthread.h>                                    教
int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */
int main(int argc, char *argv[])
{
pthread_t tid; /* the th
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_create(&tid, &attr, runner, argv[1]);
pthread_join(tid , NULL);
printf("sum = %d\n", sum);
}
```

get the default attributes

线程对应的函数

create the thread

wait for the thread to exit

```
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);
}
```

线程中止

```
sp@localhost:~/os> vim pthread.c
sp@localhost:~/os> gcc -o pthread pthread.c -pthread
sp@localhost:~/os> ./pthread 5
sum = 15
```

③等待多个子线程结束

```
#define MUM_THREADS 10
……
/* an array of threads to be joined upon */
pthread_t  workers[MUM_THREADS];
……
for (int i=0; i < MUM_THREADS; i++)
    pthread_join(workers[i], NULL);
```

5、Windows 线程（暂时不确定考不考）

4.5 隐式多线程

1、Thread Pools 线程池（了解一下概念）

①在进程开始时创建一定数量的线程，加入线程池等待

②当服务器接受到请求时，如果池中有可用线程，唤醒；否则，等待

③决定线程池内线程数量的因素：

系统 CPU 的数量、物理内存大小、并发客户请求数量的预期值

④高级线程池架构根据**使用模式**动态调整池内线程数量（机制和策略分离原则）

⑤线程池的优点：

使用现有线程而不是等待创建一个线程；

线程池限制了任何时刻可用线程的数量

将执行任务从创建任务的机制中分离出来，允许采用不同策略运行任务。（比如延迟或定期执行）

4.6 线程和进程的比较

1、

①进程 = 线程 + 资源平台

②线程没有独立的资源，共享隶属进程资源

③进程创建、切换、撤消的开销远大于线程；但是进程稳定性、安全性好

④进程绝对隔离，线程完全共享

2、进程与线程的比较

对比维度	多进程	多线程	占优
数据共享同步	共享复杂、需要用IPC；同步简单	共享简单、同步复杂	各有优势
内存、CPU	占用内存多、切换复杂CPU利用率低	占用内存少，切换简单、CPU利用率高	线程
创建、销毁、切换	复杂、速度慢	简单、速度很快	线程
编程、调试	简单	复杂	进程
可靠性	进程间不会互相影响	一个线程挂掉将导致整个进程挂掉	进程
分布式	适应于多核、多机分布式；如果一台机器不够，扩展到多台机器比较简单	适应于多核分布式	进程

3、并发模式的选择

①线程优先

强相关的处理、多核分布的、大量计算、频繁创建销毁

②进程优先

弱相关、多机分布的

③总原则：

都满足需求的情况下，选择最熟悉擅长的方式；

实际应用：进程+线程

4.7 操作系统例子

1、Windows 线程

①一对一模型

②线程包括：线程 ID、线程上下文（寄存器组、用户栈、内核栈、私有存储区域）

③**内核空间**：执行线程块、内核线程块；**用户空间**：线程环境块

2、Linux 线程（重点）

①一对一模型

②主流的线程实现方案：NPTL

③**内核态不区分**线程和进程，都使用 task_struct；**2 者区别**：一些属性的共享程度不同；**线程是轻量级进程**

④进程创建 fork：

内核调用 do_fork()，进程拥有独立的运行环境

⑤线程创建 pthread_create：

调用 clone()，来设置要共享的资源（clone()允许一个子进程共享父进程的地址空间）

创建的“进程”（线程）拥有共享的运行环境，只有栈是独立的

3、一个包含 4 个线程的进程

①**进程描述符**包含**指向 4 个不同线程的指针**

②**线程本身**再去描述它独占的资源

③Linux 中创建 4 个进程并分配 4 个普通 task_struct 结构，并指定他们共享某些资源