Technology Tarena 达内科技

Unix系统高级编程

Thread 1

DAY14

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	线程
	10:30 ~ 11:20	线程的创建与汇合
	11:30 ~ 12:20	线程的终止与取消
下午	14:00 ~ 14:50	线程属性
	15:00 ~ 15:50	基于并发线程的网络银行
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑

线程

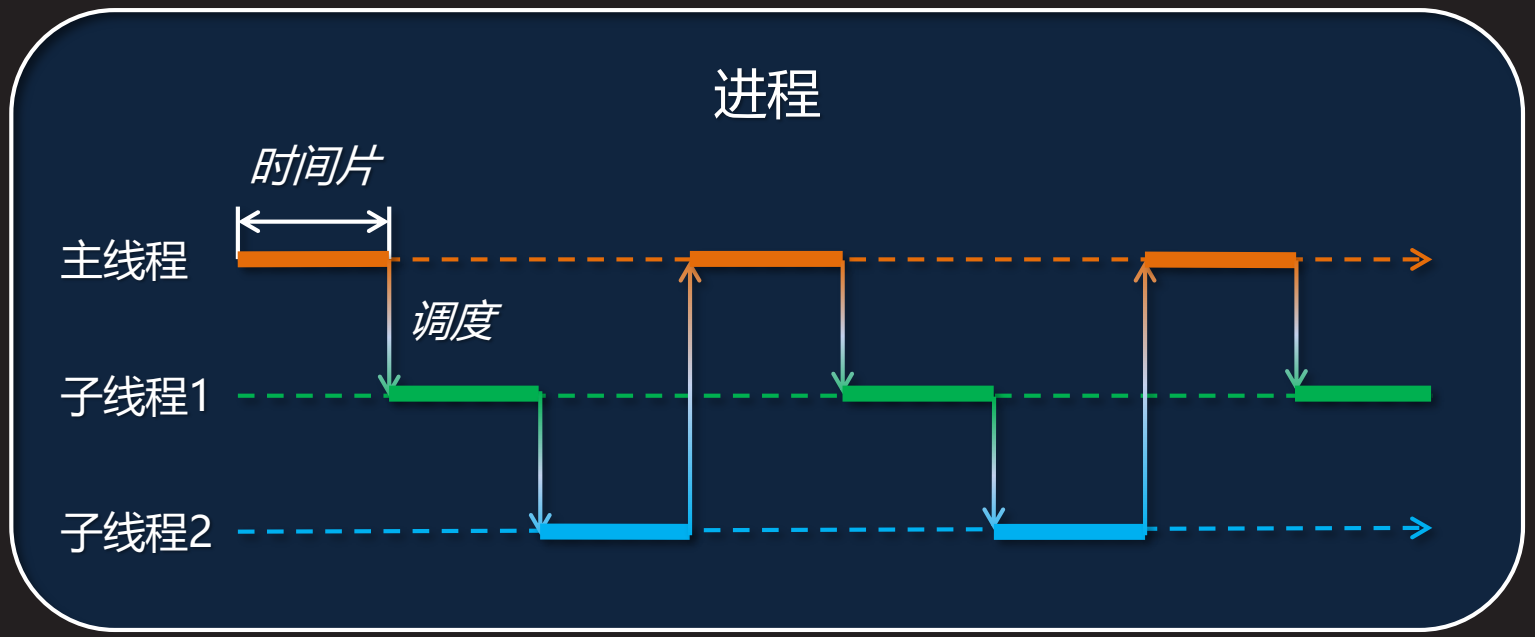
Technology Tarena 达内科技

线程的基本概念

线程的基本概念

- 线程就是程序的执行路线，即进程内部的控制序列，或者说是进程的子任务
- 一个进程可以同时拥有多个线程，即同时被系统调度的多条执行路线，但至少要有一个主线程

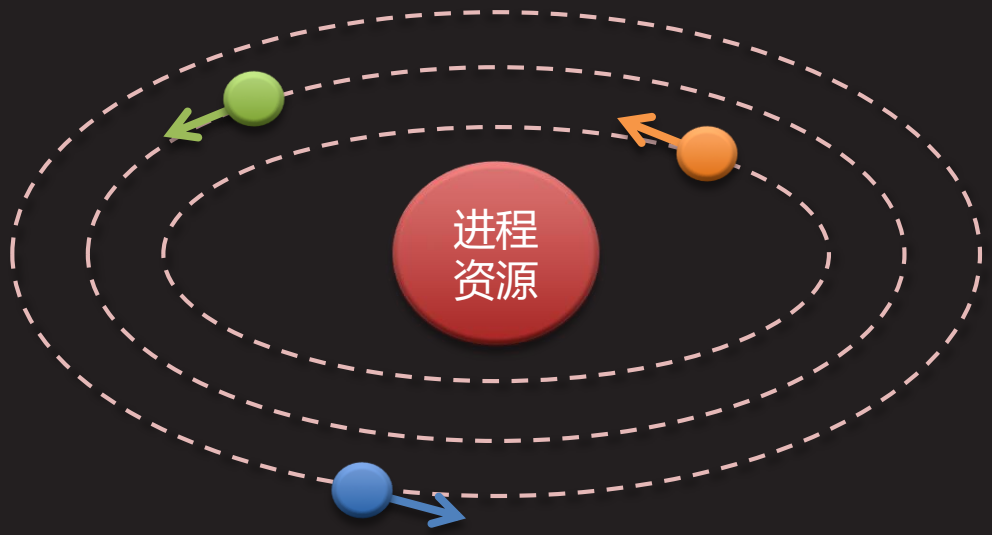
知识讲解



线程的基本概念（续1）

- 一个进程的所有线程都共享进程的代码区、数据区、堆区(注意没有栈区)、环境变量和命令行参数、文件描述符、信号处理函数、当前目录、用户ID和组ID等
- 一个进程的每个线程都拥有独立的ID、寄存器值、栈内存、调度策略和优先级、信号掩码、errno变量以及线程私有数据(Thread Specific Data, TSD)等

知识讲解



线程的基本概念（续2）

- 线程调度
  - 系统内核中专门负责线程调度的处理单元被称为调度器
  - 调度器将所有处于就绪状态(没有阻塞在任何系统调用上)的线程排成一个队列，即所谓就绪队列
  - 调度器从就绪队列中获取队首线程，为其分配一个时间片，并令处理机执行该线程，过了一段时间
    - 该线程的时间片耗尽，调度器立即中止该线程，并将其排到就绪队列尾端，接着从队首获取下一个线程
    - 该线程的时间片未耗尽，但需阻塞于某系统调用，比如等待I/O或者睡眠。调度器会中止该线程，并将其从就绪队列移至等待队列，直到其等待的条件满足后，再被移回就绪队列
  - 在低优先级线程执行期间，有高优先级线程就绪，后者会抢占前者的时间片
  - 若就绪队列为空，则系统内核进入空闲状态，直至其非空

知识讲解

线程的基本概念（续3）

- 时间片
  - 调度器分配给每个线程的时间片长短，对系统的行为和性能影响很大
    - 如果时间片过长，线程必须等待很长时间才能重新获得处理机，这就降低了整个系统运行的并行性，用户会感觉到明显的响应延迟
    - 如果时间片过短，大量时间会浪费在线程切换上，同时降低了虚拟内存的存储命中率，线程的时间局部性无法得到保证
  - 某些Unix系统倾向于为线程分配较长的时间片，希望通过扩大系统吞吐率来改善其整体表现；而另一些Unix系统则更倾向于为线程分配较短的时间片，以提升系统的交互性
  - Linux系统根据线程在不同时间的具体表现，为其动态分配时间片，在吞吐率和交互性之间寻求最佳平衡点

知识讲解

线程的基本概念（续4）

Technology Tarena 达内科技

- 处理机约束与I/O约束
  - 一些线程总是持续地消耗掉分配给它们的全部时间片，比如那些专门负责科学计算、图像处理等任务的线程。这些线程被称为处理机约束型线程。它们通常可以得到较长的时间片，通过提高虚拟内存的存储命中率，保证线程的时间局部性，以尽可能快的速度完成任务
  - 另一些线程则多数时间处于为等待资源而阻塞的状态，比如那些专门负责文件读写、网络通信或者人机交互的线程。这些线程被称为I/O约束型线程。它们通常只能得到较短的时间片，因为它们在发出I/O请求并阻塞于内核资源之前，只会运行很短的时间
  - 另一方面，调度器又会适度降低处理机约束型线程的优先级，同时提高I/O约束型线程的优先级，于其间寻求平衡

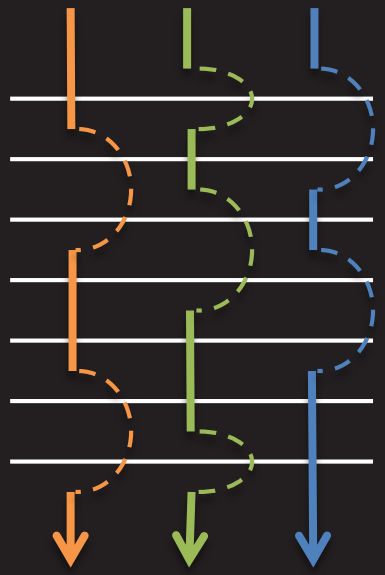
知识讲解

Technology Tarena 达内科技

线程的基本特点

线程的基本特点

- 线程是进程的一个实体，可作为系统独立调度和分派的基本单位
- 线程有不同的状态，系统提供了多种线程控制原语，如创建、终止、取消等等
- 线程可以使用的大部分资源都是隶属于进程的，即使是在特定线程中动态分配的资源，也同样为进程所拥有
- 一个进程中可以有多个线程并发地运行。它们可以执行相同的代码，也可以执行不同的代码



知识讲解

线程的基本特点（续1）

- 同一个进程的多个线程都在同一个地址空间内活动，因此相较于进程，线程的系统开销小，任务切换快
- 进程空间内的代码和数据对于该进程的每个线程而言都是共享的。因此同一个进程的不同线程之间不存在通信问题，当然也就不需要类似IPC的通信机制
- 线程之间虽然不存在通信问题但是存在冲突问题。同样是因为数据共享，当一个进程的多个线程“同时”访问一份数据时，线程间的冲突可能造成逻辑甚至系统错误
- 线程之间存在优先级的差异。即使低优先级线程的时间片尚未耗尽，只要高优先级线程处于就绪状态，就会立即抢夺低优先级线程手中的处理机

知识讲解

Technology Tarena 达内科技

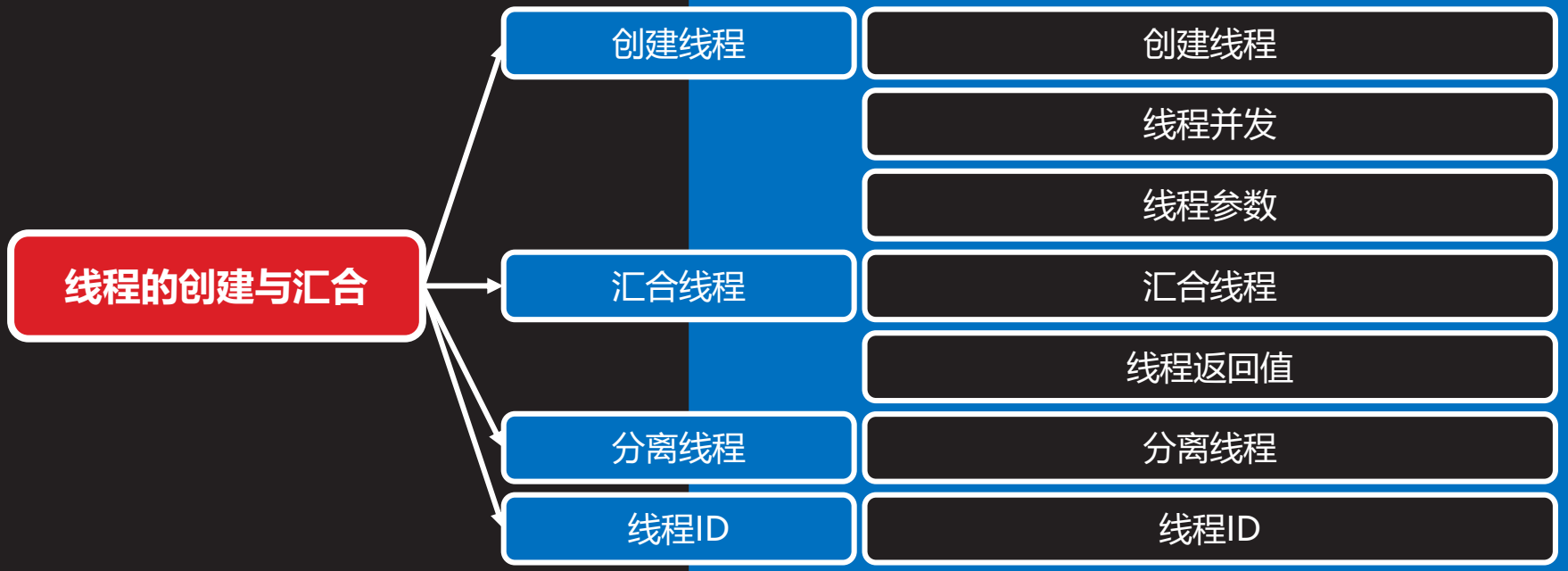
POSIX线程

POSIX线程

- 早期Unix厂商各自提供私有的线程库版本，无论是接口还是实现，差异都非常大，代码移植非常困难
- IEEE POSIX 1003.1c (1995)标准，定义了统一的线程编程接口，遵循该标准的线程实现被统称为POSIX线程，即pthread
- 使用pthread需要包含一个头文件：pthread.h
  同时连一个共享库：libpthread.so
  - #include <pthread.h>
  - gcc ... -lpthread

知识讲解

线程的创建与汇合


线程的创建与汇合
创建线程
汇合线程
分离线程
线程ID
创建线程
线程并发
线程参数
汇合线程
线程返回值
分离线程
线程ID

Technology Tarena 达内科技

创建线程

创建线程

Technology Tarena 达内科技

- 创建新线程

```
#include <pthread.h>

int pthread_create (pthread_t* thread,
    const pthread_attr_t* attr,
    void* (*start_routine) (void*), void* arg);
```

成功返回0，失败返回错误码

知识讲解

- ***thread***：输出线程ID。pthread_t即unsigned long int
- ***attr***：线程属性，NULL表示缺省属性。 pthread_attr_t可能是整型也可能是结构体，因实现而异
- ***start_routine***：线程过程函数指针。参数和返回值的类型都是void*。启动线程其实就是调用一个函数，只不过是在一个独立的线程中调用的，函数一旦返回，线程即告结束

创建线程（续1）

- 创建新线程
  - ***arg***：传递给线程过程函数的参数。线程过程函数的调用者是系统内核，因此需要预先将参数存储到系统内核中
- 在pthread.h头文件中声明的函数，通常以直接返回错误码的方式表示失败，而非返回-1并设置errno
- main函数可以被视为主线程的线程过程函数。main函数一旦返回，主线程即告结束。主线程一旦结束，进程即告结束。进程一旦结束，其所有的子线程统统结束
- 应设法保证在线程过程函数执行期间，传递给它的参数***arg***所指向的目标持久有效

知识讲解

创建线程（续2）

- 例如
  - void* start_routine (void* arg) {
    ...
    return NULL;
    }
  - pthread_t thread;
    int error = pthread_create (&thread, NULL,
    start_routine, NULL);
    if (error) {
    printf ("pthread_create: %s\n", strerror (error));
    exit (EXIT_FAILURE);
    }

创建线程

【参见：create.c】

课堂练习

- 创建线程

线程并发

知识讲解

- pthread_create函数本身并不调用线程过程函数，而是在系统内核中开启独立的线程，并立即返回，在该线程中执行线程过程函数中的代码
- pthread_create函数返回时，所指定的线程过程函数可能尚未执行，也可能正在执行，甚至可能已经执行完毕
- 如果主线程先于子线程结束，则由于进程的结束，所有子线程都会被直接终止
- 主线程和通过pthread_create函数创建的多个子线程，在时间上“同时”运行，如果不附加任何同步条件，则它们每一个执行步骤的先后顺序是完全无法确定的，这就叫做自由并发

线程并发

【参见：concur.c】

课堂练习

- 线程并发

线程参数

- 传递给线程过程函数的参数是一个泛型指针void*，它可以指向任何类型的数据：基本类型变量、结构体型变量或者数组型变量等等，但必须保证在线程过程函数执行期间，该指针所指向的目标变量持久有效，直到线程过程函数不再使用它为止
- 调用pthread_create函数的代码在用户空间，线程过程函数的代码也在用户空间，但偏偏创建线程的动作由系统内核完成。因此传递给线程过程函数的参数也不得不经由系统内核传递给线程过程函数。pthread_create函数的***arg***参数负责将线程过程函数的参数带入系统内核

知识讲解

线程参数

【参见：arg.c】

课堂练习

- 线程参数

Technology Tarena 达内科技

汇合线程

汇合线程

- 等待线程终止并与之汇合，同时回收该线程的相关资源

```
#include <pthread.h>

int pthread_join (pthread_t thread, void** retval);
```

成功返回0，失败返回错误码

  - ***thread***：线程ID
  - ***retval***：输出线程过程函数的返回值。为了获得线程过程函数返回的一级泛型指针，可以同样定义一个一级泛型指针，并将其地址通过pthread_join函数的二级泛型指针参数***retval***传入该函数，并由该函数在***thread***线程终止以后，将其线程过程函数返回的一级指针填入该二级指针的目标

知识讲解

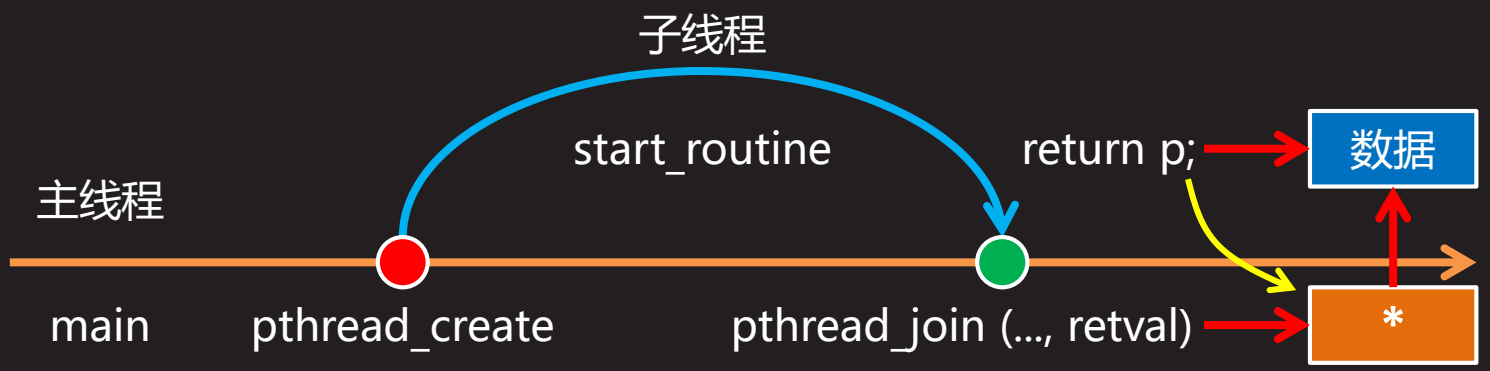
汇合线程（续1）

- 例如
  - void* start_routine (void* arg) {
    double* rs = (double*)arg;
    *rs = PI * *rs * *rs;
    return NULL;
    }
  - double rs = 10.0;
    pthread_t thread;
    pthread_create (&thread, NULL, start_routine, &rs);
    pthread_join (thread, NULL);
    printf ("圆面积：%g\n", rs);

线程返回值

- 从线程过程函数中返回值的方法
  - 线程过程函数将所需返回的内容放在一块内存中，返回该内存的地址，同时保证这块内存，在函数返回即线程结束以后依然有效
  - 若***retval***参数非NULL，则pthread_join函数将线程过程函数所返回的指针，拷贝到该参数所指向的内存中
  - 若线程过程函数所返回的指针指向动态分配的内存，则还需保证在用过该内存之后释放之

知识讲解



线程返回值（续1）

- 例如
  - void* start_routine (void* arg) {
    double r = *(double*)arg;
    double* s = malloc (sizeof (double));
    *s = PI * r * r;
    return s; }
  - double r = 10.0;
    pthread_t thread;
    pthread_create (&thread, NULL, start_routine, &r);
    double* s;
    pthread_join (thread, (void**)&s);
    printf ("圆面积：%g\n", *s);
    free (s);

知识讲解

汇合线程

【参见：ret.c】

课堂练习

- 汇合线程

Technology Tarena 达内科技

分离线程

分离线程

- 使指定线程进入分离状态

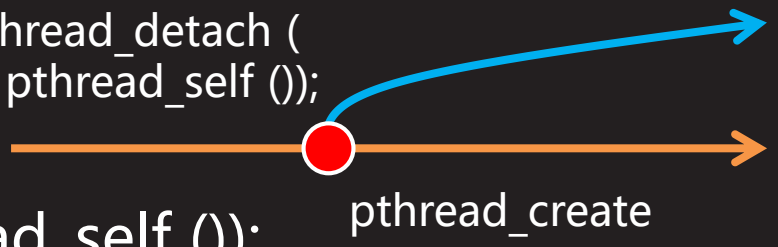
```
#include <pthread.h>

int pthread_detach (pthread_t thread);

                          成功返回0，失败返回错误码
```

  - *thread*：线程ID

- pthread_detach函数使*thread*线程进入分离状态(DETACHED)。处于分离状态的线程一旦终止，其线程资源即被系统自动回收。处于分离状态的线程不能被pthread_join函数汇合
- 例如
  - pthread_detach (pthread_self ());



知识讲解

分离线程

【参见：detach.c】

课堂练习

- 分离线程

Technology Tarena 达内科技

线程ID

线程ID

- 获取调用线程的ID

```
#include <pthread.h>

pthread_t pthread_self (void);
```

成功返回调用线程的ID，不会失败

- 例如
  - pthread_t thread = pthread_self ();
    printf ("线程ID：%ul\n", thread);

知识讲解

线程ID（续1）

- 判断线程ID是否相等

```
#include <pthread.h>

int pthread_equal (pthread_t t1, pthread_t t2);

        若t1和t2所表示的线程ID相等，则返回非零，否则返回0
```

  - *t1*：线程ID
  - *t2*：线程ID

- 并非所有实现的pthread_t都是unsigned long int类型，有些可能是结构体类型，不能用“==”判断其是否相等
- 例如
  - printf (%d\n", pthread_equal (t1, t2));

知识讲解

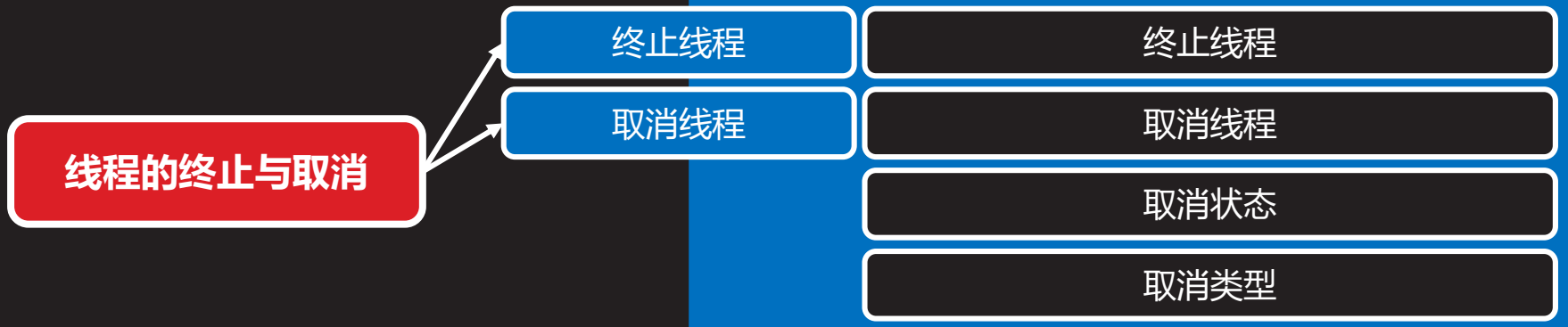
线程ID

【参见：equal.c】

课堂练习

- 线程ID

线程的终止与取消



Technology Tarena 达内科技

终止线程

终止线程

- 令调用线程终止运行

```
#include <pthread.h>

void pthread_exit (void* retval);
```

无返回值

  - retval：相当于线程过程函数的返回值

- 在线程过程函数或者被线程过程函数直接或间接调用的函数中，调用pthread_exit函数，其效果都与在线程过程函数中执行return语句的效果一样——终止调用线程
- 注意，在任何线程中调用exit函数，被终止的都是进程。当然随着进程的终止，隶属于该进程的包括调用线程在内的所有线程也都一并终止

终止线程（续1）

- 例如
  - void circle_area (double r) {
    double* s = malloc (sizeof (double));
    *s = PI * r * r;
    pthread_exit (s); }
  - void* start_routine (void* arg) {
    circle_area (*(double*)arg); return NULL; }
  - double r = 10.0;
    pthread_t thread;
    pthread_create (&thread, NULL, start_routine, &r);
    double* s;
    pthread_join (thread, (void**)&s);
    printf ("圆面积：%g\n", *s); free (s);

知识讲解

终止线程

【参见：exit.c】

课堂练习

- 终止线程

Technology Tarena 达内科技

取消线程

取消线程

- 向指定线程发送取消请求

```
#include <pthread.h>

int pthread_cancel (pthread_t thread);

                                成功返回0，失败返回错误码
```

  - *thread*：线程ID
- 该函数只是向线程发出取消请求，并不等待线程终止
- 缺省情况下，线程在收到取消请求以后，并不会立即终止，而是仍继续运行，直到其达到某个取消点。在取消点处，线程检查其自身是否已被取消，若是则立即终止
- 当线程调用一些特定函数时，取消点会出现

知识讲解

取消状态

- 设置调用线程的取消状态

```
#include <pthread.h>

int pthread_setcancelstate (int state, int* oldstate);
```

成功返回0，失败返回错误码

– ***state***：取消状态，可取以下值

`PTHREAD_CANCEL_ENABLE` - 接受取消请求(缺省)

`PTHREAD_CANCEL_DISABLE` - 忽略取消请求

– ***oldstate***：输出原取消状态，可取NULL

知识讲解

取消类型

- 设置调用线程的取消类型

```
#include <pthread.h>

int pthread_setcanceltype (int type, int* oldtype);
```

成功返回0，失败返回错误码

– ***type***：取消类型，可取以下值

PTHREAD_CANCEL_DEFERRED - 延迟取消(缺省)
被取消线程在接收到取消请求之后并不立即终止，而是一直等到执行了特定的函数(取消点)之后再终止

PTHREAD_CANCEL_ASYNCHRONOUS - 异步取消
被取消线程可以在任意时刻终止，而不是非得遇到取消点

– ***oldtype***：输出原取消类型，可取NULL

知识讲解

取消线程

【参见：cancel.c】

课堂练习

- 取消线程

线程属性

Technology Tarena 达内科技

线程属性结构

线程属性结构

- 如前所述，创建线程的函数pthread_create的第二个参数***attr***即线程属性，传空指针表示使用缺省属性
  - int pthread_create (pthread_t* ***thread***,
    const pthread_attr_t* ***attr***,
    void* (****start_routine***) (void*), void* ***arg***);
- 线程属性用pthread_attr_t结构体表示，包括以下成员
  - int detachstate; // 分离状态
    - PTHREAD_CREATE_JOINABLE - 可汇合线程(缺省)
    - PTHREAD_CREATE_DETACHED - 分离线程
  - int scope; // 竞争范围
    - PTHREAD_SCOPE_SYSTEM - 系统范围竞争
    - PTHREAD_SCOPE_PROCESS - 进程范围竞争(Linux不支持)

知识讲解

线程属性结构（续1）

- 线程属性用pthread_attr_t结构体表示，包括以下成员
  - int inheritsched; // 继承调度
    - PTHREAD_INHERIT_SCHED - 调度属性自父线程继承(缺省)
    - PTHREAD_EXPLICIT_SCHED - 调度属性由后两个成员确定
  - int schedpolicy; // 调度策略
    - SCHED_FIFO - 先进先出策略
      没有时间片。一个FIFO线程会持续运行，直到阻塞或有高优先级线程就绪。当FIFO线程阻塞时，系统将其移出就绪队列，待其恢复时再加到同优先级就绪队列的末尾。当FIFO线程被高优先级线程抢占时，它在就绪队列中的位置不变。因此一旦高优先级线程终止或阻塞，被抢占的FIFO线程会立即运行
    - SCHED_RR - 轮转策略
      给每个RR线程分配一个时间片，一但RR线程的时间片耗尽，即将其移至就绪队列末尾

知识讲解

线程属性结构（续2）

Technology Tarena 达内科技

- 线程属性用pthread_attr_t结构体表示，包括以下成员
  - int schedpolicy; // 调度策略
    - SCHED_OTHER - 普通策略(缺省)
      静态优先级为0。任何就绪的FIFO线程或RR线程，都会抢占此类线程
  - struct sched_param schedparam; // 调度参数
    - struct sched_param {
      int sched_priority; // 静态优先级
      };
      静态优先级仅对实时调度SCHED_FIFO/SCHED_RR有意义。此值越大优先级越高。调用sched_get_priority_max和sched_get_priority_min函数，可获得当前系统所支持的最大和最小静态优先级

知识讲解

线程属性结构（续3）

Technology Tarena 达内科技

- 线程属性用pthread_attr_t结构体表示，包括以下成员
  - size_t guardsize; // 栈尾警戒区大小(以字节为单位)
    - 缺省为一页，即4096字节
  - void* stackaddr; // 栈区起始地址
    - 线程栈区的起始地址被定义为线程栈内存的最低地址，但这并不必然就是线程栈的开始位置。某些处理器架构，栈是从高地址向低地址方向伸展的，那么所谓栈区起始地址实际上是栈区的结尾位置而非开始位置
  - size_t stacksize; // 栈区大小(以字节为单位)
- 并非所有的实现都将pthread_attr_t类型定义为上述结构体。因此设置或者获取线程属性时最好不要手工读写该结构体，而是调用专门的pthread_attr_set/get函数设置/获取具体的属性项

知识讲解

Technology Tarena 达内科技

设置线程属性

设置线程属性

- 初始化线程属性结构(成功返回0，失败返回错误码)
  - int pthread_attr_init (pthread_attr_t* ***attr***);
- 设置具体线程属性项(成功返回0，失败返回错误码)
  - int pthread_attr_setdetachstate (pthread_attr_t* ***attr***, int ***detachstate***); // 分离状态
  - int pthread_attr_setscope (pthread_attr_t* ***attr***, int ***scope***); // 竞争范围
  - int pthread_attr_setinheritsched (pthread_attr_t* ***attr***, int ***inheritsched***); // 继承调度
  - int pthread_attr_setschedpolicy (pthread_attr_t* ***attr***, int ***policy***); // 调度策略
  - int pthread_attr_setschedparam (pthread_attr_t* ***attr***, const struct sched_param* ***param***); // 调度参数

设置线程属性（续1）

- 设置具体线程属性项(成功返回0，失败返回错误码)
  - int pthread_attr_setguardsize (pthread_attr_t* ***attr***, size_t ***guardsize***); // 栈尾警戒区大小
  - int pthread_attr_setstackaddr (pthread_attr_t* ***attr***, void* ***stackaddr***); // 栈区起始地址(已过时，不推荐使用)
  - int pthread_attr_setstacksize (pthread_attr_t* ***attr***, size_t ***stacksize***); // 栈区大小
  - int pthread_attr_setstack (pthread_attr_t* ***attr***, void* ***stackaddr***, size_t ***stacksize***); // 栈区起始地址和大小
- 用设置好的线程属性创建线程
  - int pthread_create (pthread_t* ***thread***, const pthread_attr_t* ***attr***, void* (*****start_routine***) (void*), void* ***arg***);

知识讲解

设置线程属性（续2）

- 销毁线程属性结构(成功返回0，失败返回错误码)
  - int pthread_attr_destroy (pthread_attr_t* ***attr***);
- 例如
  - pthread_attr_t attr;
    pthread_attr_init (&attr));
    pthread_attr_setdetachstate (&attr,
    PTHREAD_CREATE_DETACHED);
    pthread_t thread;
    pthread_create (&thread, &attr, start_routine, NULL);
    pthread_attr_destroy (&attr);

知识讲解

Technology Tarena 达内科技

获取线程属性

获取线程属性

- 获取线程的属性结构(成功返回0，失败返回错误码)
  - int pthread_getattr_np (pthread_t ***thread***, pthread_attr_t* ***attr***);
- 获取具体线程属性项(成功返回0，失败返回错误码)
  - int pthread_attr_getdetachstate (pthread_attr_t* ***attr***, int* ***detachstate***); // 分离状态
  - int pthread_attr_getscope (pthread_attr_t* ***attr***, int* ***scope***); // 竞争范围
  - int pthread_attr_getinheritsched (pthread_attr_t* ***attr***, int* ***inheritsched***); // 继承调度
  - int pthread_attr_getschedpolicy (pthread_attr_t* ***attr***, int* ***policy***); // 调度策略

知识讲解

获取线程属性（续1）

- 获取具体线程属性项(成功返回0，失败返回错误码)
  - int pthread_attr_getschedparam (pthread_attr_t* ***attr***, struct sched_param* ***param***); // 调度参数
  - int pthread_attr_getguardsize (pthread_attr_t* ***attr***, size_t* ***guardsize***); // 栈尾警戒区大小
  - int pthread_attr_getstackaddr (pthread_attr_t* ***attr***, void** ***stackaddr***); // 栈区起始地址(已过时，不推荐使用)
  - int pthread_attr_getstacksize (pthread_attr_t* ***attr***, size_t* ***stacksize***); // 栈区大小
  - int pthread_attr_getstack (pthread_attr_t* ***attr***, void** ***stackaddr***, size_t* ***stacksize***); // 栈区起始地址和大小
- 销毁线程属性结构(成功返回0，失败返回错误码)
  - int pthread_attr_destroy (pthread_attr_t* ***attr***);

知识讲解

获取线程属性（续2）

- 例如
  - pthread_attr_t attr;
    pthread_getattr_np (pthread_self (), &attr);
    int detachstate;
    pthread_attr_getdetachstate (attr, &detachstate);
    if (detachstate == PTHREAD_CREATE_JOINABLE)
        printf ("可汇合线程\n");
    else
    if (detachstate == PTHREAD_CREATE_DETACHED)
        printf ("分离线程\n");
    else
        printf ("未知\n");
    pthread_attr_destroy (&attr);

线程属性

【参见：attr.c】

课堂练习

- 线程属性

线程的静态优先级

【参见：priority.c】

课堂练习

- 线程的静态优先级

基于并发线程的网络银行

Technology Tarena 达内科技

需求分析

需求分析

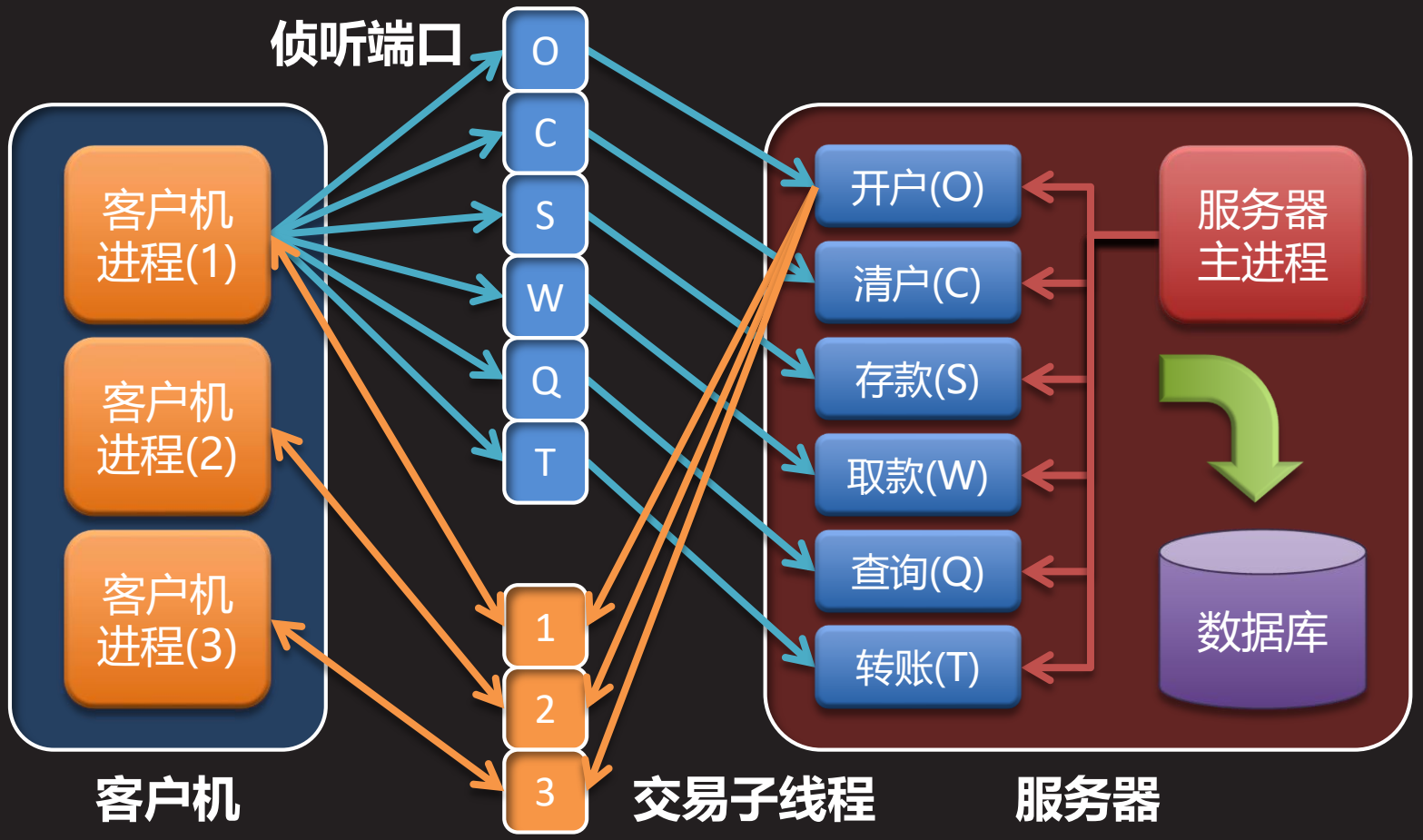
知识讲解

- 基于并发线程的网络银行，包括客户机和服务器两个组成部分。其中客户机负责提供用户界面，接收用户的输入，并向其提供输出，服务器负责后台业务逻辑的处理
- 每一种银行业务皆被实现为一个独立的进程，谓之业务服务。服务器主进程负责各业务服务进程的启动与终止
- 客户机与各业务服务之间通过TCP协议在同一台或不同机器上通信，故谓之网络银行。客户机根据所需要的业务种类连接相应的业务服务。后者负责创建专门与该客户机通信的交易子线程，接收客户机的请求、完成业务处理，并向客户机返回响应，直至关闭连接，终止线程
- 业务数据需要在服务器端持久化，采用文件系统数据库

Technology Tarena 达内科技

概要设计

概要设计



- 每个业务服务均以相应的业务标识作为侦听端口号
- 每个业务服务都用独立的子线程处理客户机的业务

Technology Tarena 达内科技

开发计划

开发计划

1. 修改网络通信模块：network.h、network.c
2. 删除太平间信号处理模块：mortuary.h、mortuary.c
3. 修改各业务服务模块：open.c、close.c、save.c、withdraw.c、query.c、transfer.c
4. 修改构建脚本：makefile
5. 链接、运行、测试

知识讲解

基于并发线程的网络银行

【参见：bank/】

课堂练习

- 基于并发线程的网络银行

Technology Tarena 达内科技

总结和答疑