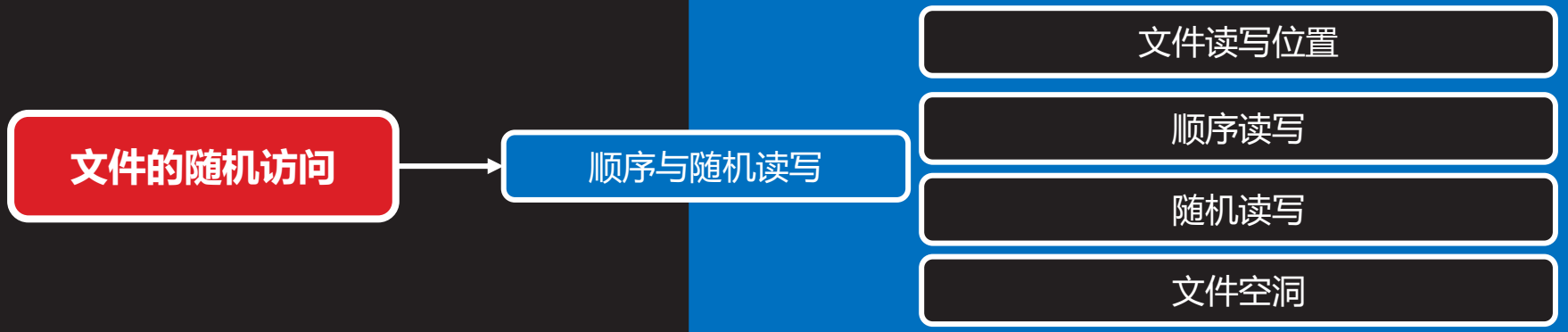
Technology Tarena 达内科技

Unix系统高级编程

随机访问和文件同步 Unit09

文件的随机访问



Technology Tarena 达内科技

顺序与随机读写

文件读写位置

- 每个打开的文件都有一个与其相关的文件读写位置保存在文件表项中，用以记录从文件头开始计算的字节偏移
- 文件读写位置通常是一个非负的整数，用off_t类型表示，在32位系统上被定义为long int，而在64位系统上则被定义为long long int
- 打开一个文件时，除非指定了O_APPEND标志，否则文件读写位置一律被设为0，即文件首字节的位置

知识讲解

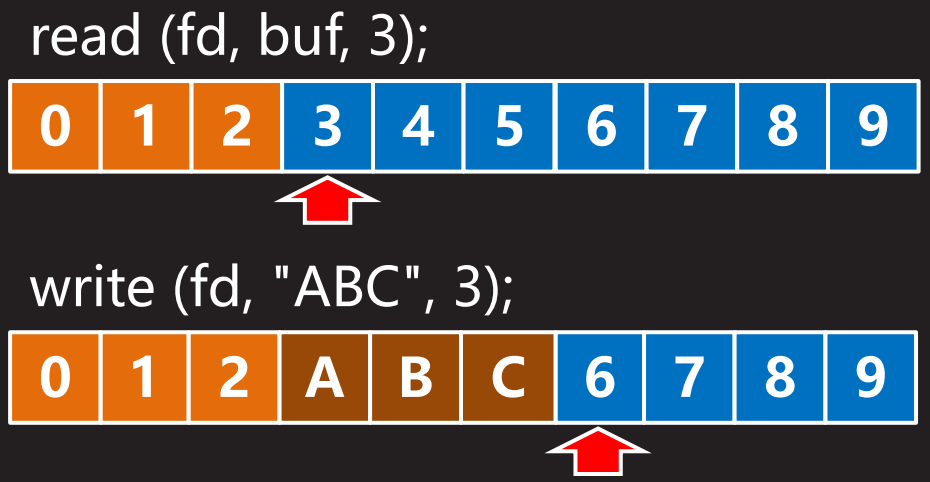
creat ("write.txt", 0644);

write (fd, "0123456789", 10);

顺序读写

- 每一次读写操作都从当前的文件读写位置开始，并根据所读写的字节数，同步增加文件读写位置，为下一次读写做好准备
- 因为文件读写位置是保存在文件表项而不是v节点中的，因此通过多次打开同一个文件得到多个文件描述符，各自拥有各自的文件读写位置

知识讲解



随机读写

- 人为调整文件读写位置

```
#include <unistd.h>

off_t lseek (int fd, off_t offset, int whence);

                    成功返回调整后的文件读写位置，失败返回-1
```

  - *fd*：文件描述符
  - *offset*：文件读写位置相对于*whence*参数的偏移量

知识讲解

随机读写（续1）

- 人为调整文件读写位置
  - ***whence***：根据***offset***参数计算文件读写位置的起点，可取以下值

    SEEK_SET - 从文件头(文件的第一个字节)开始

    SEEK_CUR - 从当前位置(上次读写的最后一个字节的下一个位置)开始

    SEEK_END - 从文件尾(文件最后一个字节的下一个位置)开始
- lseek函数的功能仅仅是修改保存在文件表项中的文件读写位置，并不实际引发任何I/O动作

知识讲解

随机读写（续2）

- 例如
  - lseek (fd, -7, SEEK_CUR); // 从当前位置向文件头偏移7字节
  - lseek (fd, 0, SEEK_CUR); // 返回当前文件读写位置
  - lseek (fd, 0, SEEK_END); // 返回文件总字节数

知识讲解

read (fd, buf, 3);

lseek (fd, 3, SEEK_CUR);

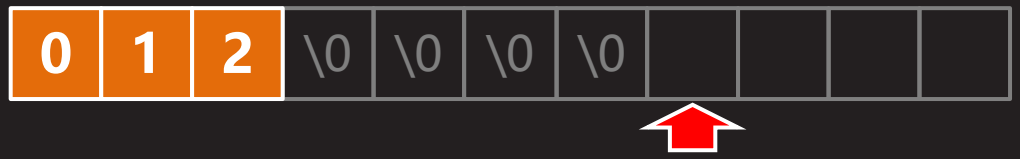
文件空洞

- 可以通过lseek函数将文件读写位置设到文件尾之后
- 在超过文件尾的位置上写入数据，将在文件中形成空洞，位于文件中但没有被写过的字节都被设为0
- 文件空洞不占用磁盘空间，但被计算在文件大小之内

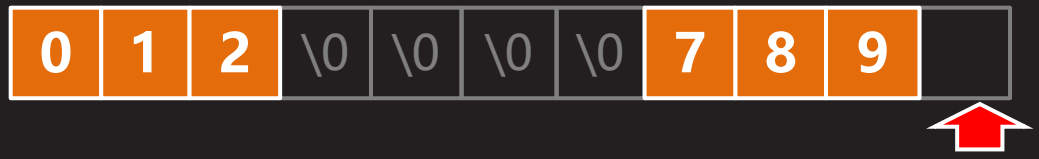
知识讲解

write (fd, "012", 3);

lseek (fd, 4, SEEK_CUR);



write (fd, "789", 3);



顺序与随机读写

【参见：seek.c】

课堂练习

- 顺序与随机读写

复制文件描述符

Technology Tarena 达内科技

dup

dup

- 复制文件描述符表项

```
#include <unistd.h>

int dup (int oldfd);

                成功返回目标文件描述符，失败返回-1
```

  - *oldfd*：源文件描述符

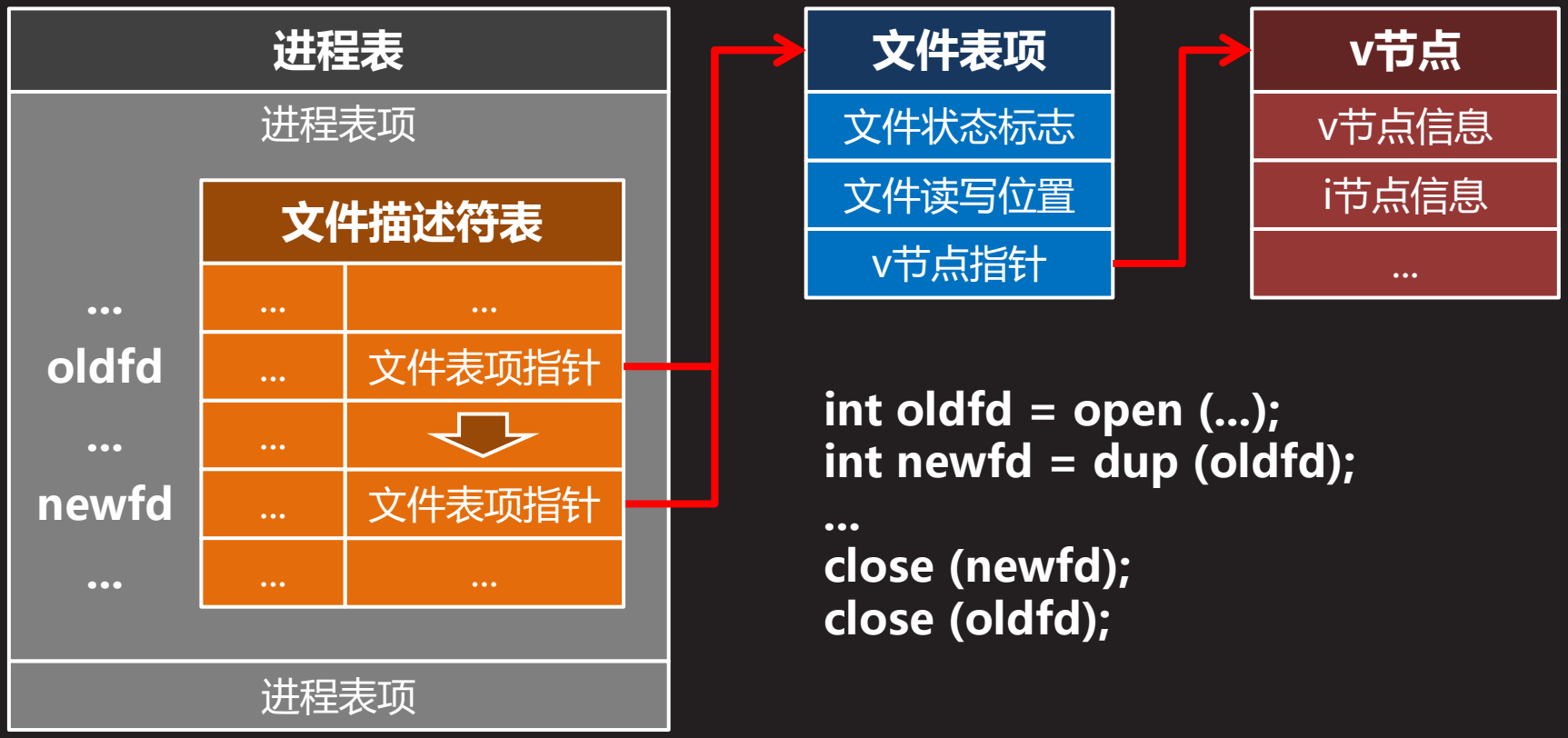
- dup函数将*oldfd*参数所对应的文件描述符表项复制到文件描述符表第一个空闲项中，同时返回该表项所对应的文件描述符
- dup函数返回的文件描述符一定是调用进程当前未使用的最小文件描述符

知识讲解

dup（续1）

- dup函数只复制文件描述符表项，不复制文件表项和v节点，因此该函数所返回的文件描述符可以看做是参数文件描述符*oldfd*的副本，它们标识同一个文件表项

知识讲解




```
int oldfd = open (...);
int newfd = dup (oldfd);
...
close (newfd);
close (oldfd);
```

dup（续2）

- 注意，当关闭文件时，即使是由dup函数产生的文件描述符副本，也应该通过close函数关闭，因为只有当关联于一个文件表项的所有文件描述符都被关闭了，该文件表项才会被销毁，类似地，也只有当关联于一个v节点的所有文件表项都被销毁了，v节点才会被从内存中删除，因此从资源合理利用的角度讲，凡是明确不再继续使用的文件描述符，都应该尽可能及时地用close函数关闭
- dup函数究竟会把***oldfd***参数所对应的文件描述符表项，复制到文件描述符表的什么位置，程序员是无法控制的，这完全由调用该函数时文件描述符表的使用情况决定，因此对该函数的返回值做任何约束性假设都是不严谨的

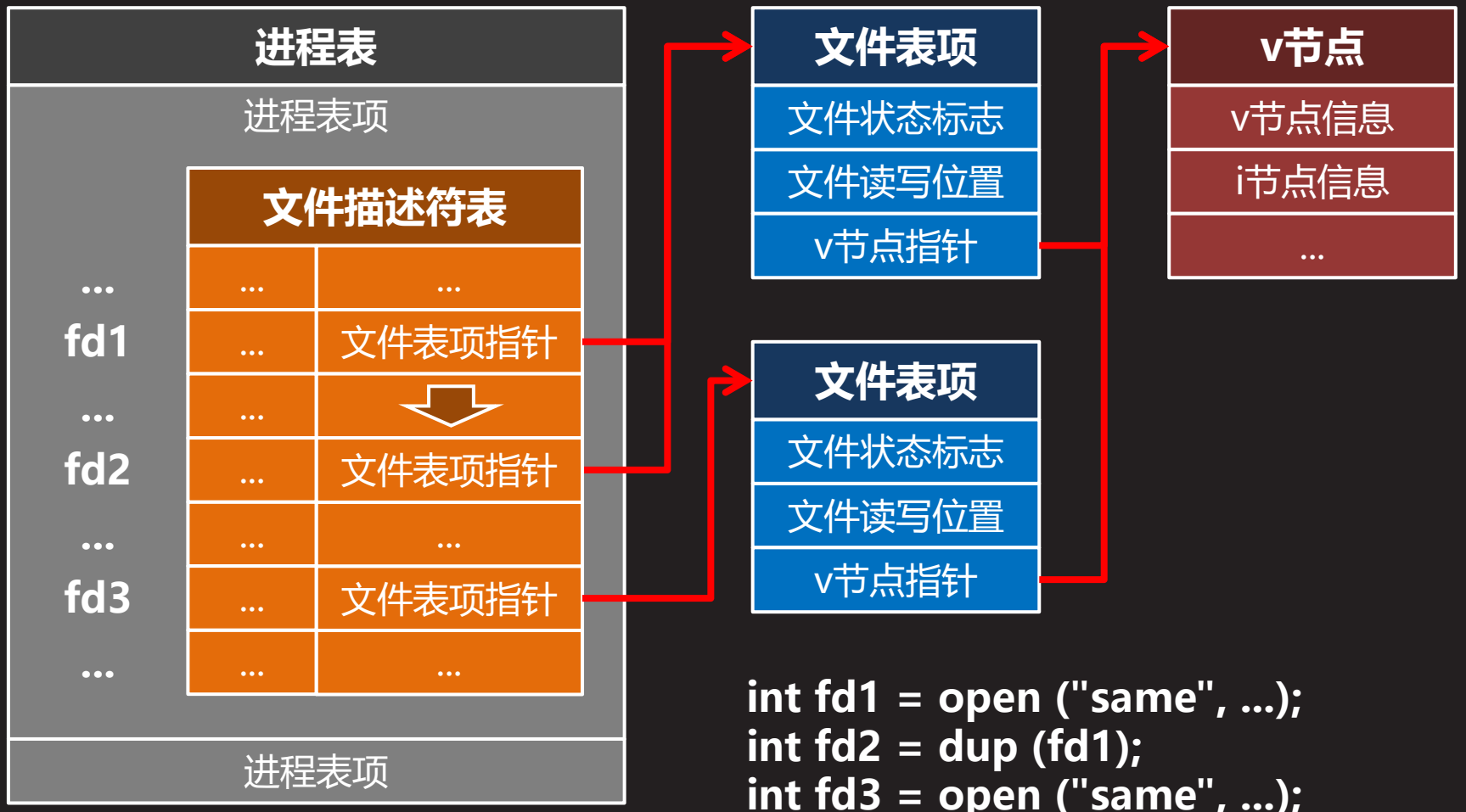
知识讲解

dup（续3）

- 由dup函数返回的文件描述符与作为参数传递给该函数的文件描述符标识的是同一个文件表项，而文件读写位置是保存在文件表项而非文件描述符表项中的，因此通过这些文件描述符中的任何一个，对文件进行读写或随机访问，都会影响通过其它文件描述符操作的文件读写位置。这与多次通过open函数打开同一个文件不同

知识讲解

dup（续4）




```
int fd1 = open ("same", ...);
int fd2 = dup (fd1);
int fd3 = open ("same", ...);
```

知识讲解

Technology Tarena 达内科技

dup2

dup2

- 复制文件描述符表项到指定位置

```
#include <unistd.h>

int dup2 (int oldfd, int newfd);

                成功返回目标文件描述符，失败返回-1
```

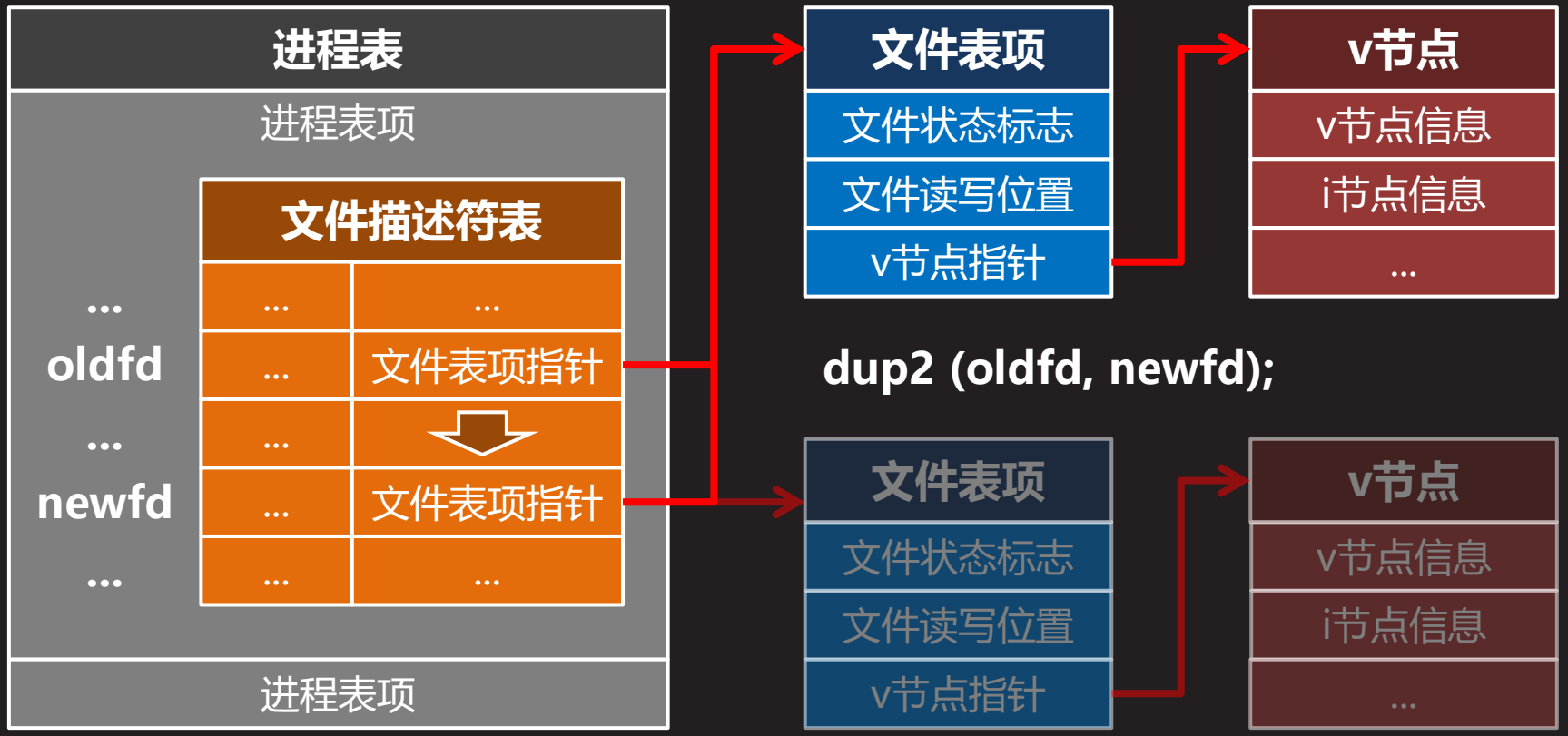
  - ***oldfd***：源文件描述符
  - ***newfd***：目标文件描述符

- dup2函数的功能与dup函数几乎完全一样，唯一的不同就是允许调用者通过***newfd***参数指定目标文件描述符，正常情况下该函数的返回值应该与***newfd***参数的值相等

知识讲解

dup2（续1）

- dup2函数在复制由*oldfd*参数所标识的源文件描述符表项时，会首先检查由*newfd*参数所标识的目标文件描述符表项是否空闲，若空闲则直接将前者复制给后者，否则会先将目标文件描述符*newfd*关闭，再行复制



知识讲解

复制文件描述符

【参见：dup.c】

课堂练习

- 复制文件描述符

系统与标准I/O

Technology Tarena 达内科技

系统I/O

系统I/O

- 当系统调用函数被执行时，需要在用户态和内核态之间来回切换，因此频繁执行系统调用函数会严重影响性能
  - unsigned int i;
    for (i = 0; i < 10000000; ++i)
        write (fd, &i, sizeof (i));

知识讲解

$ time sysio

```
real    0m33.385s
user    0m0.752s
sys     0m32.586s
```

系统I/O

【参见：sysio.c】

课堂练习

- 系统I/O

Technology Tarena 达内科技

标准I/O

标准I/O

- 标准库做了必要的优化，内部维护一个缓冲区，只在满足特定条件时才将缓冲区与系统内核同步，借此降低执行系统调用的频率，减少进程在用户态和内核态之间来回切换的次数，提高运行性能
  - unsigned int i;
    for (i = 0; i < 10000000; ++i)
        fwrite (&i, sizeof (i), 1, fp);

$ time stdio

```
real    0m13.012s
user    0m0.176s
sys     0m12.805s
```

知识讲解

标准I/O

【参见：stdio.c】

课堂练习

- 标准I/O

文件同步

Technology Tarena 达内科技

文件同步

文件同步

- 写缓冲与延迟写
  - 当一个运行在用户空间的进程发起write系统调用时，系统内核进行几项检查，然后直接将通过参数传入的数据块拷贝到一个被称为写缓冲的缓冲区中，随即返回调用进程
  - 稍后，运行在后台的系统内核收集所有这样的“脏”缓冲区，将它们按照一定的顺序排入写队列，并逐一写入磁盘，这个过程被称为回写(writeback)，而这种将回写操作延迟执行的策略被称为延迟写
  - 延迟写一方面使运行在用户空间的调用进程不必等待写磁盘动作(通常较慢)的实际完成，提高其运行速度，另一方面使系统内核得以将实际的写磁盘工作推迟到相对空闲的时候完成，且以批量方式集中处理，提高内核的工作效率

知识讲解

文件同步（续1）

- 延迟写的潜在风险
  - write系统调用的成功返回，仅表示通过其参数传入的数据块，已被成功地拷贝到由系统内核负责维护的写缓冲中，而此后任何在回写过程中发生错误，如物理磁盘驱动器故障等，都不会被报告给发出写请求的进程
  - 如果在“脏”缓冲区中的数据被实际写入磁盘之前，系统发生了某种不可预期的异常，如硬件故障、突然掉电等，写缓冲中的数据将永远失去被写入磁盘的机会，造成数据丢失(即失步)，这对许多关键业务系统而言无疑是致命的

知识讲解

文件同步（续2）

- 延迟写的潜在风险
  - 为了降低延迟写的风险，保证写缓冲中的数据变化适时地被同步到磁盘，系统建立了一个写缓冲最大时效机制，并保证所有的“脏”缓冲区在它们超过给定时效前能够被强制同步到磁盘设备。系统管理员可以通过/proc/sys/vm/dirty_expire_centisecs来配置这个值，该值以厘秒(1厘秒=0.01秒=10毫秒)为单位
  - 如果对系统提供的上述风险防范机制仍然心存疑虑，或者希望能够严格控制数据被写入磁盘的时间，那么就必须考虑采用同步I/O的手段，以牺牲性能为代价，换取更高的可靠性和精确性。这对那些关键性的业务数据显得尤其重要

文件同步（续3）

- 同步文件内容和元数据

```
#include <unistd.h>

int fsync (int fd);
```

成功返回0，失败返回-1

  - *fd*：需要同步的文件描述符
- 调用fsync函数可以保证*fd*参数所标识文件的“脏”数据立即被回写到磁盘上
- 文件描述符*fd*必须是以写方式打开的
- 该函数不仅回写文件的内容数据，诸如文件大小、时间戳等保存在i节点中的元数据也一并被回写

知识讲解

文件同步（续4）

- fsync函数在磁盘驱动器确认所有与特定文件相关的“脏”数据都被成功回写之前不会返回，因此调用进程一旦从该函数返回成功，就完全有理由确信*fd*参数所标识的文件已经同步完成
- 注意，现代大多数磁盘设备也是带有缓冲区的，fsync函数的同步也只是把数据同步到磁盘设备的缓冲区里，至于这些数据是否真的被保存到物理介质上了，fsync函数是不可能知道的(硬盘可能会撒谎，它通知系统内核缓冲数据已经写到磁盘上了，但实际上它们仍在磁盘缓存中)

知识讲解

文件同步（续5）

- 只同步文件内容不同步元数据

```
#include <unistd.h>

int fdatasync (int fd);
```

成功返回0，失败返回-1

  - *fd*：需要同步的文件描述符

- fdatasync函数与fsync函数的功能几乎完全一样，唯一的区别在于它只回写文件的内容数据，而诸如文件大小、时间戳等保存在i节点中的元数据，该函数不保证同步，因此fdatasync函数的执行速度比fsync函数要快一些

文件同步（续6）

- 同步所有文件的内容和元数据

```
#include <unistd.h>

void sync (void);
```

永远成功，无返回值

  - 标准中的sync函数将系统中所有存在“脏”数据的写缓冲统统排入写队列，随即返回，并不等待写磁盘操作的完成
  - Linux的sync函数一定要等到所有写缓冲中的数据，既包括文件的内容数据也包括文件的元数据，都被实际写入磁盘设备后才返回
  - 在一个繁忙的系统上，一次对sync函数的调用，其耗时可能要长达数分钟之久

知识讲解

文件同步（续7）

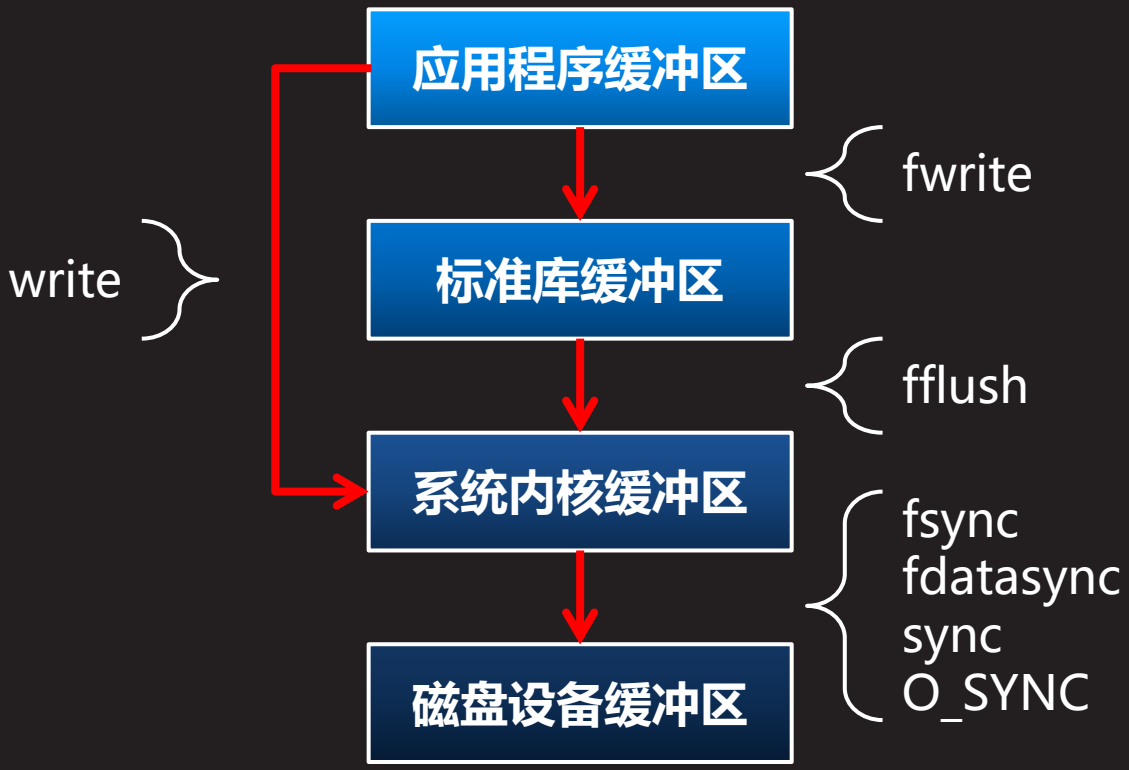
- 如果在打开文件时使用了O_SYNC标志，则所有在这个文件描述符上的写操作都是同步的，即使不调用fsync、fdatasync或者sync函数
- 以同步模式打开的文件，会在I/O等待上消耗大量的运行时间，使用O_SYNC的进程比不使用O_SYNC的进程要慢一两个数量级，这显然是一种无奈之举
- 通过fsync或fdatasync函数实现同步的情况要好得多，毕竟只有那些关键性的业务数据才需要强制同步，总体开销比使用O_SYNC要小得多
- 读操作从来都是同步的，什么时候要什么时候读，要多少读多少，提前读显然不象延迟写那么有价值

知识讲解

文件同步（续8）

- 缓冲！缓冲！缓冲！
  - 从应用程序到标准库，再到系统内核，最后到硬件设备，每一层都会维护自己的缓冲区，以改善系统的整体性能

知识讲解



Technology Tarena 达内科技

总结和答疑