Technology Tarena 达内科技

Unix系统高级编程

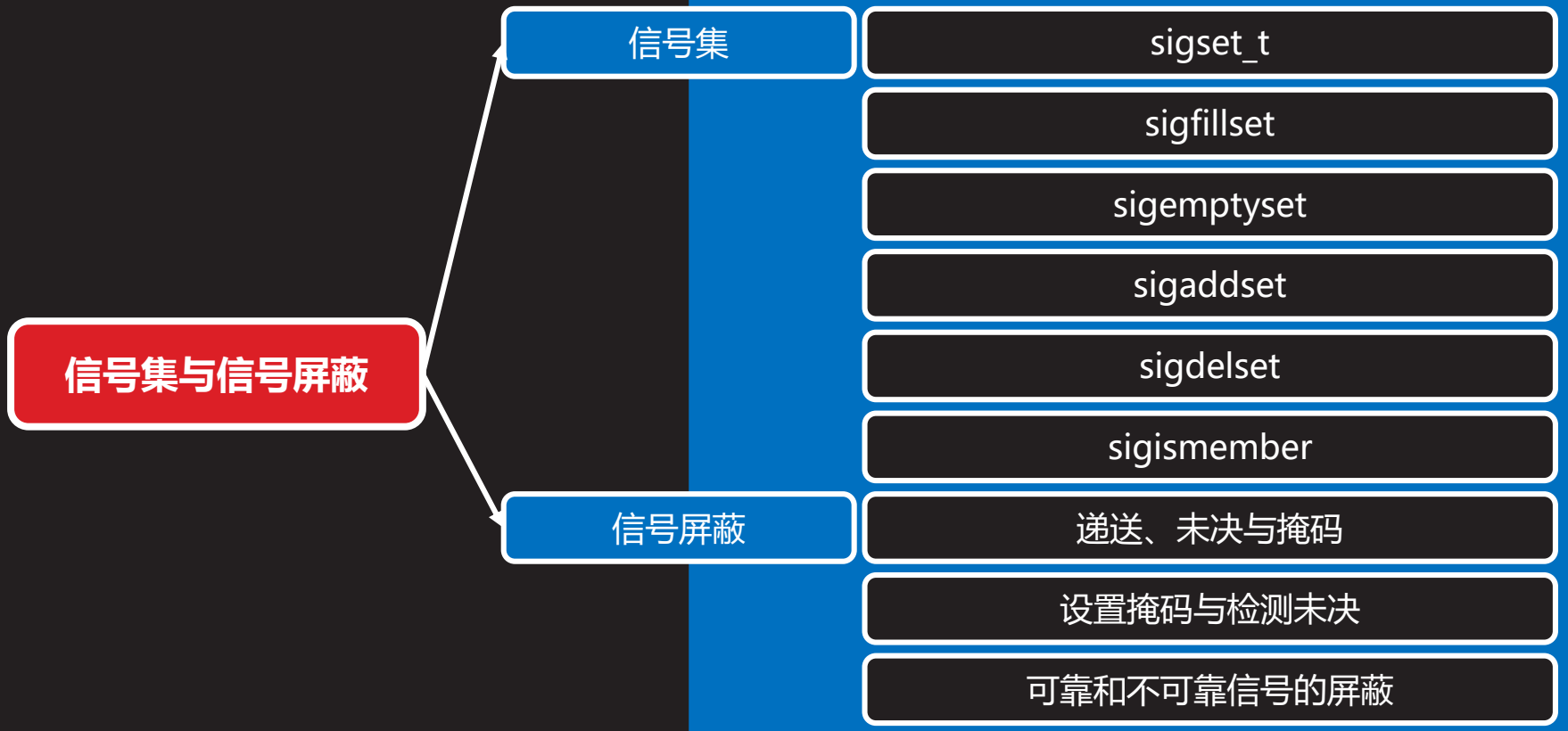
Signal 2

DAY11

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	信号集与信号屏蔽
	10:30 ~ 11:20	
	11:30 ~ 12:20	信号处理与发送
下午	14:00 ~ 14:50	定时器
	15:00 ~ 15:50	进程间通信
	16:00 ~ 16:50	管道
	17:00 ~ 17:30	总结和答疑

信号集与信号屏蔽


信号集与信号屏蔽
信号集
sigset_t
sigfillset
sigemptyset
sigaddset
sigdelset
sigismember
信号屏蔽
递送、未决与掩码
设置掩码与检测未决
可靠和不可靠信号的屏蔽

Technology Tarena 达内科技

信号集

sigset_t

Technology Tarena 达内科技

- 多个信号组成的信号集合谓之信号集
- 系统内核用sigset_t类型表示信号集
- sigset_t类型是一个结构体，但该结构体中只有一个成员，是一个包含32个元素的整数数组
  - 在<sigset.h>中有如下类型定义

```
#define _SIGSET_NWORDS (1024 /
    (8 * sizeof (unsigned long int)))
typedef struct {
    unsigned long int __val[_SIGSET_NWORDS];
}   __sigset_t;
```

  - 在<signal.h>中又被定义为

```
typedef __sigset_t sigset_t;
```

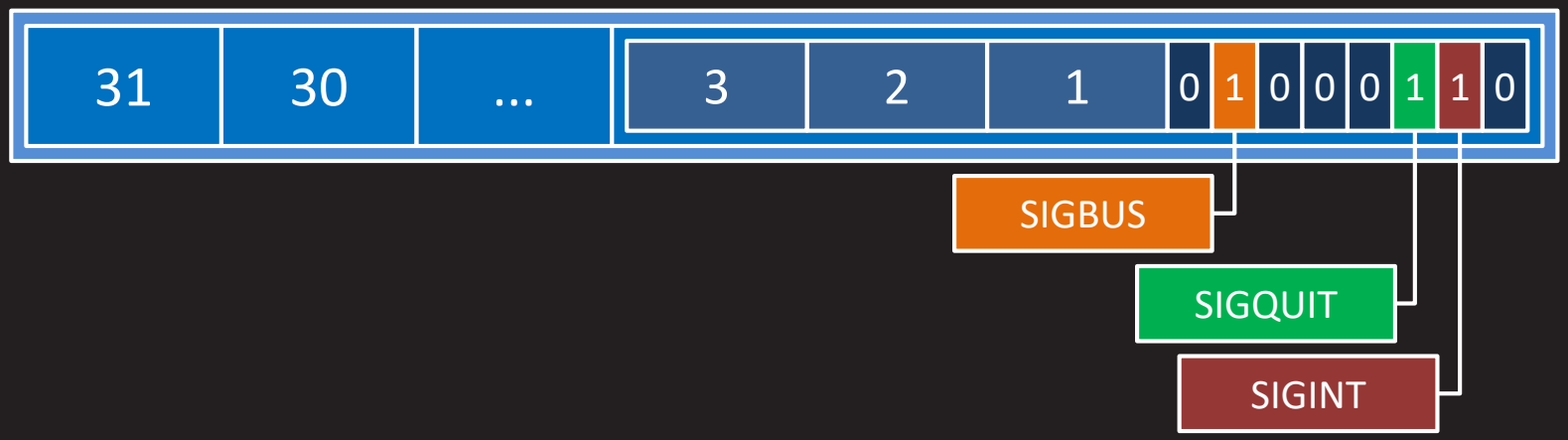
知识讲解

sigset_t（续1）

Technology Tarena 达内科技

- 可以把sigset_t类型看成一个由1024个二进制位组成的大整数
  - 其中的每一位对应一个信号，其实目前远没有那么多信号
  - 某位为1就表示信号集中有此信号，反之为0就是无此信号
  - 当需要同时操作多个信号时，常以sigset_t作为函数的参数或返回值的类型

知识讲解



sigfillset

- 填满信号集，即将信号集的全部信号位置1

```
#include <signal.h>

int sigfillset (sigset_t* sigset);
```

成功返回0，失败返回-1

  - *sigset*：信号集
- 例如
  - sigset_t sigset;
    if (sigfillset (&sigset) == -1) {
        perror ("sigfillset");
        exit (EXIT_FAILURE);
    }

知识讲解

sigemptyset

- 清空信号集，即将信号集的全部信号位清0

```
#include <signal.h>

int sigemptyset (sigset_t* sigset);
```

成功返回0，失败返回-1

  - *sigset*：信号集

- 例如
  - sigset_t sigset;

```
if (sigemptyset (&sigset) == -1) {
    perror ("sigemptyset");
    exit (EXIT_FAILURE);
}
```

知识讲解

sigaddset

Technology Tarena 达内科技

- 加入信号，即将信号集中与指定信号编号对应的信号位置1

```
#include <signal.h>

int sigaddset (sigset_t* sigset, int signum);
```

成功返回0，失败返回-1

  - *sigset*：信号集
  - *signum*：信号编号

- 例如
  -
```
if (sigaddset (&sigset, SIGINT) {
    perror ("sigaddset");
    exit (EXIT_FAILURE);
}
```

知识讲解

sigdelset

Technology Tarena 达内科技

- 删除信号，即将信号集中与指定信号编号对应的信号位清0

```
#include <signal.h>

int sigdelset (sigset_t* sigset, int signum);
```

成功返回0，失败返回-1

  - *sigset*：信号集
  - *signum*：信号编号
- 例如
  - if (sigdelset (&sigset, SIGINT) {
    perror ("sigdelset");
    exit (EXIT_FAILURE);
    }

知识讲解

sigismember

Technology Tarena 达内科技

- 判断信号集中是否有某信号，即检查信号集中与指定信号编号对应的信号位是否为1

```
#include <signal.h>

int sigismember (const sigset_t* sigset, int signum);
```

有则返回1，没有返回0，失败返回-1

  - *sigset*：信号集
  - *signum*：信号编号
- 例如
  - if (sigismember (&sigset, SIGINT) == 1)
    printf ("信号集中有SIGINT信号\n");

知识讲解

信号集

【参见：sigset.c】

课堂练习

- 信号集

Technology Tarena 达内科技

信号屏蔽

递送、未决与掩码

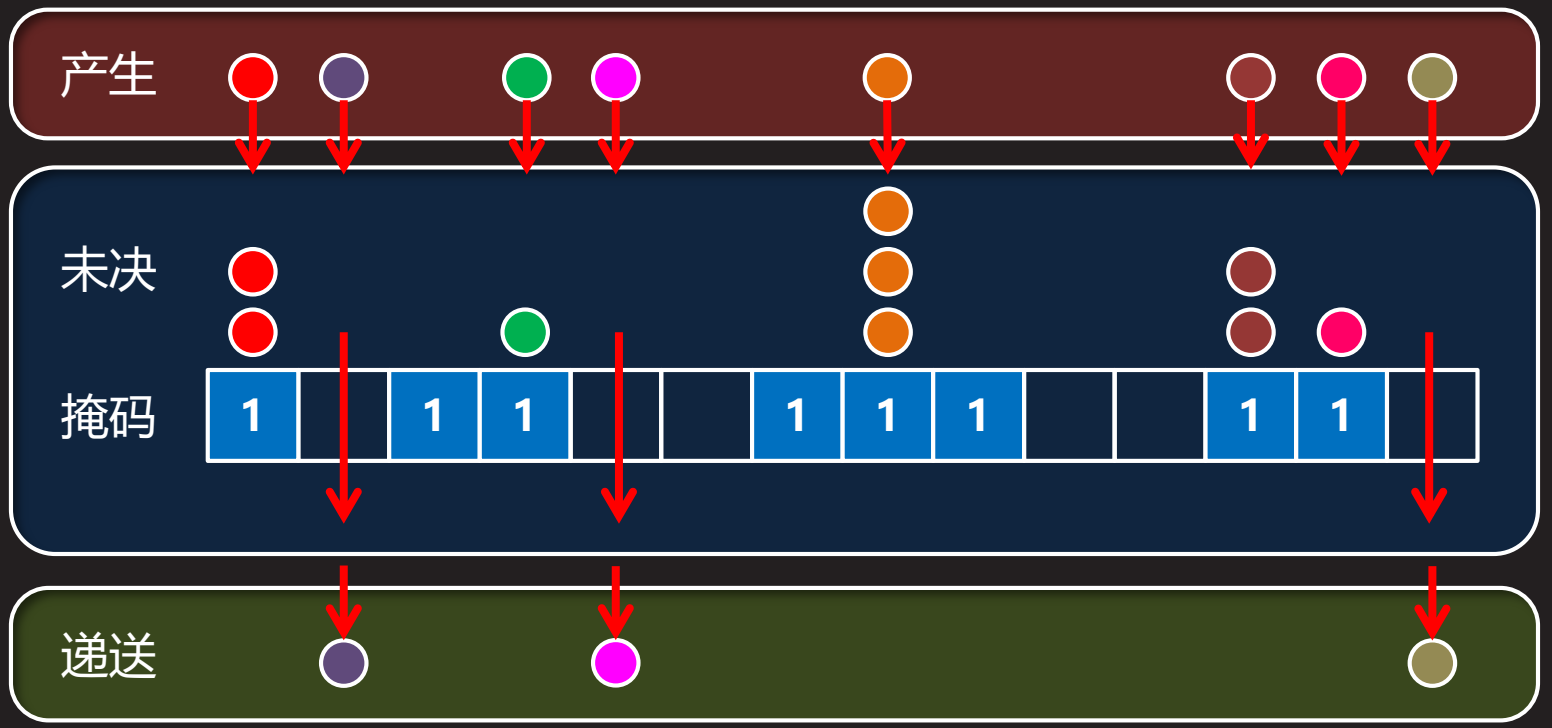
Technology Tarena 达内科技

- 当信号产生时，系统内核会在其所维护的进程表中，为特定的进程设置一个与该信号相对应的标志位，这个过程就叫做递送(delivery)
- 信号从产生到完成递送之间存在一定的时间间隔，处于这段时间间隔中的信号状态称为未决(pending)
- 每个进程都有一个信号掩码(signal mask)，它实际上是一个信号集，位于该信号集中的信号一旦产生，并不会被递送给相应的进程，而是会被阻塞(block)在未决状态
- 在信号处理函数执行期间，这个正在被处理的信号总是处于信号掩码中，如果又有该信号产生，则会被阻塞，直到上一个针对该信号的处理过程结束以后才会被递送

递送、未决与掩码（续1）

Technology Tarena 达内科技

- 当进程正在执行类似更新数据库这样的敏感任务时，可能不希望被某些信号中断。这时可以通过信号掩码暂时屏蔽而非忽略掉这些信号，使其一旦产生即被阻塞于未决状态，待特定任务完成后，再回过头来处理这些信号



设置掩码与检测未决

- 设置调用进程的信号掩码

```
#include <signal.h>

int sigprocmask (int how, const sigset_t* sigset,
    sigset_t* oldset);
```

成功返回0，失败返回-1

  - *how*：修改信号掩码的方式，可取以下值

    `SIG_BLOCK` - 将*sigset*中的信号加入当前信号掩码

    `SIG_UNBLOCK` - 从当前信号掩码中删除*sigset*中的信号

    `SIG_SETMASK` - 把*sigset*设置成当前信号掩码

知识讲解

设置掩码与检测未决（续1）

Technology Tarena 达内科技

- 设置调用进程的信号掩码
  - ***sigset***：信号集，取NULL则忽略此参数
  - ***oldset***：输出原信号掩码，取NULL则忽略此参数
- 例如
  - sigset_t sigset;
    sigemptyset (&sigset);
    sigaddset (&sigset, SIGINT);
    sigaddset (&sigset, SIGQUIT);
    sigset_t oldset;
    if (sigprocmask (SIG_SETMASK, &sigset,
    &oldset) == -1) {
    perror ("sigprocmask");
    exit (EXIT_FAILURE); }

知识讲解

设置掩码与检测未决（续2）

Technology Tarena 达内科技

- 获取调用进程的未决信号集

```
#include <signal.h>

int sigpending (sigset_t* sigset);
```

成功返回0，失败返回-1

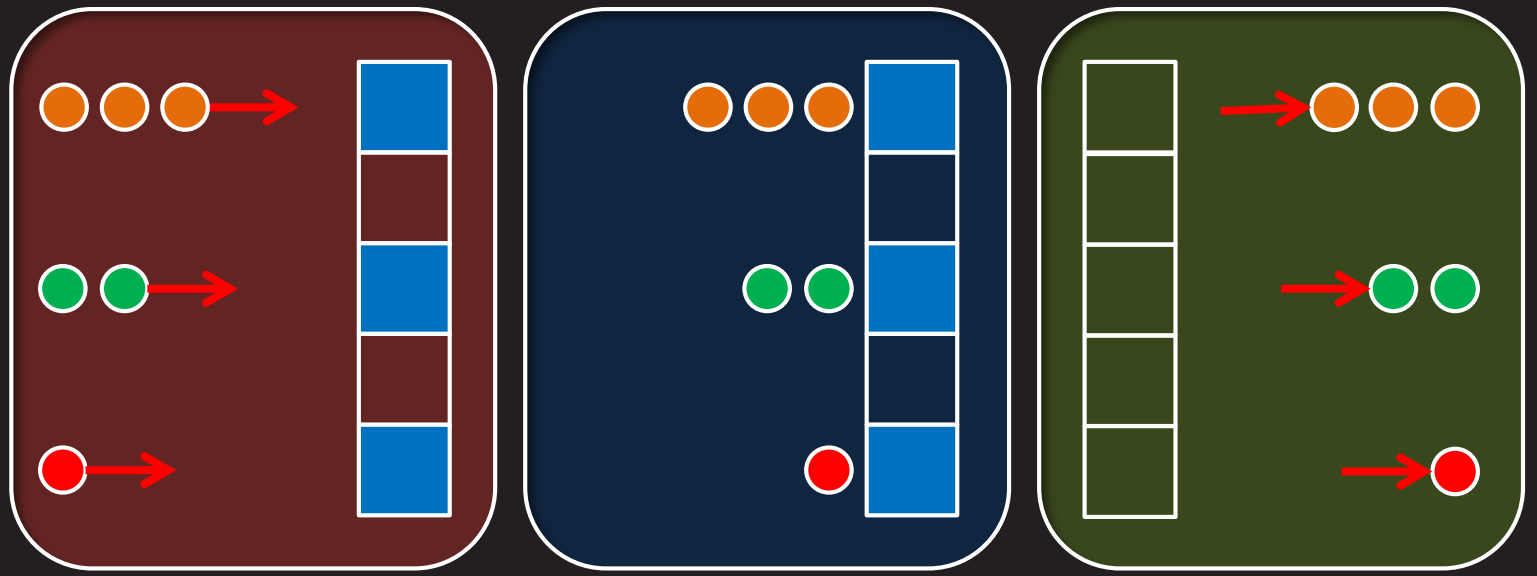
  - sigset：输出未决信号集
- 例如
  -

```
sigset_t sigset;
if (sigpending (&sigset) == -1) {
    perror ("sigpending"); exit (EXIT_FAILURE); }
if (sigismember (&sigset, SIGINT) == 1)
    printf ("SIGINT信号未决\n");
```

知识讲解

可靠和不可靠信号的屏蔽

- 对于可靠信号，通过sigprocmask函数设置信号掩码以后，每种被屏蔽信号中的每个信号都会被阻塞，并按先后顺序排队，一旦解除屏蔽，这些信号会被依次递送



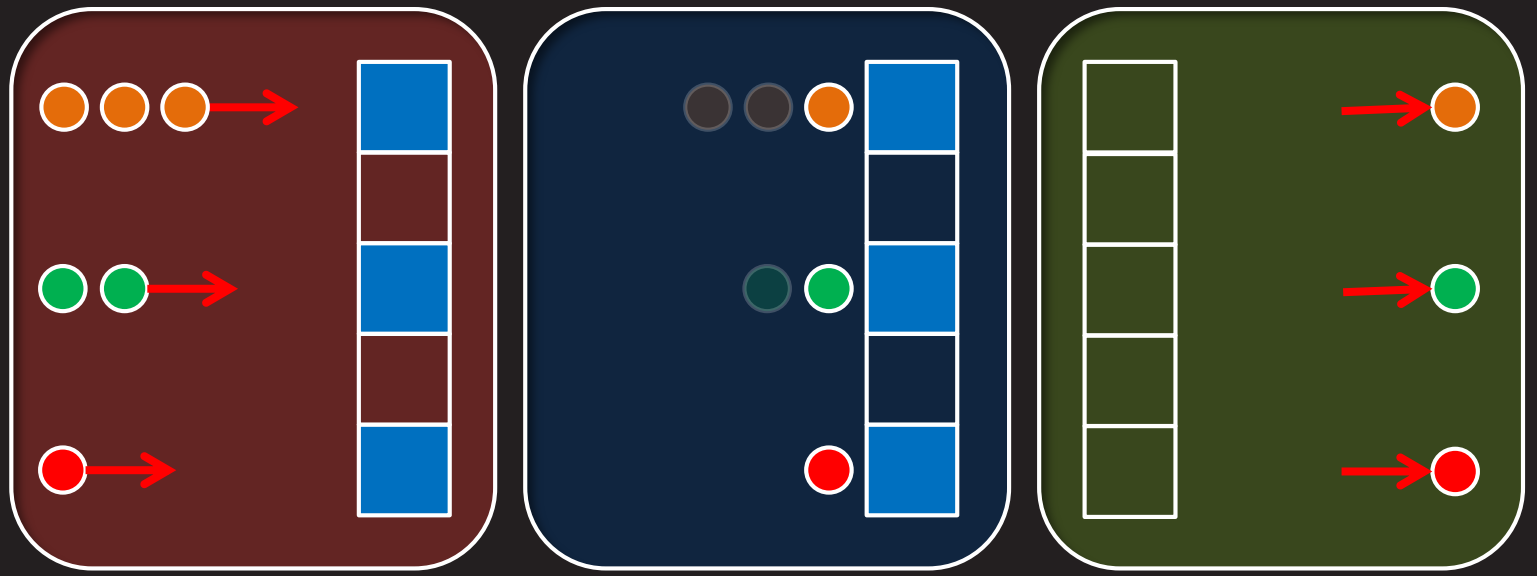
Technology Tarena 达内科技

可靠和不可靠信号的屏蔽（续1）

Technology Tarena 达内科技

- 对于不可靠信号，通过sigprocmask函数设置信号掩码以后，每种被屏蔽信号中只有第一个会被阻塞，并在解除屏蔽后被递送，其余的则全部丢失

知识讲解



Technology Tarena 达内科技

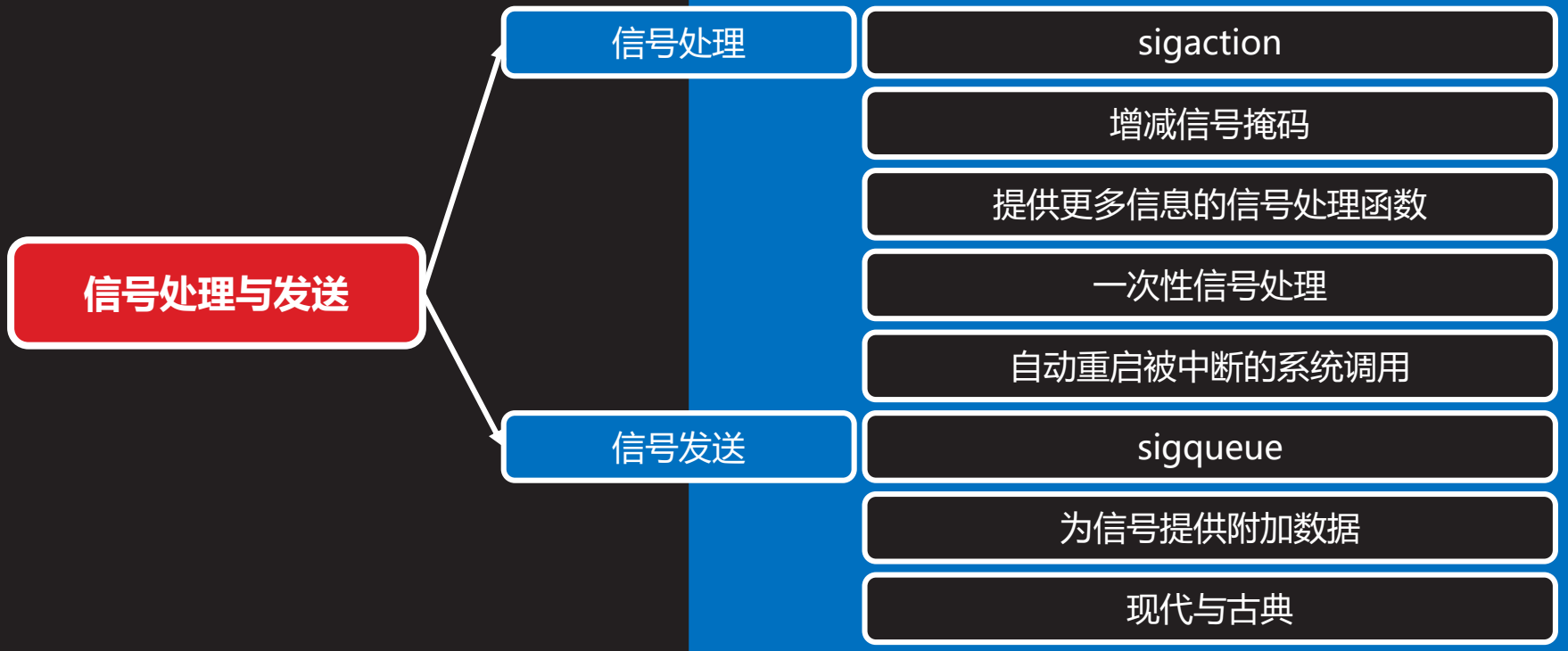
信号屏蔽

【参见：sigmask.c】

课堂练习

- 信号屏蔽

信号处理与发送


信号处理与发送
信号处理
信号发送
sigaction
增减信号掩码
提供更多信息的信号处理函数
一次性信号处理
自动重启被中断的系统调用
sigqueue
为信号提供附加数据
现代与古典

Technology Tarena 达内科技

信号处理

sigaction

- 设置针对特定信号的响应行为

```
#include <signal.h>

int sigaction (int signum, const struct sigaction* sigact,
    struct sigaction* oldact);

                                成功返回0，失败返回-1
```

  - *signum*：信号编号
  - *sigact*：信号行为
  - *oldact*：输出原信号行为，可置NULL
- 当*signum*信号被递送时，按*sigact*所描述的行为响应之
- 若*oldact*非NULL，则通过该参数输出原来的响应行为

知识讲解

Technology Tarena 达内科技

sigaction（续1）

Technology Tarena 达内科技

- sigaction函数通过信号行为结构体类型sigaction来描述对一个信号的响应行为
  - struct sigaction {

```
// 旧风格信号处理函数指针
void (*sa_handler) (int);
// 新风格信号处理函数指针
void (*sa_sigaction) (int, siginfo_t*, void*);
// 信号处理期间的附加信号掩码
sigset_t sa_mask;
// 信号处理标志
int sa_flags;
// 预留项，目前置NULL
void (*sa_restorer) (void); };
```

知识讲解

增减信号掩码

Technology Tarena 达内科技

- 缺省情况下，在信号处理函数的执行过程中，会自动屏蔽这个正在被处理的信号，而对于其它信号则不会屏蔽
- 通过sigaction::sa_mask成员可以人为指定，在信号处理函数执行期间，除正在被处理的信号外，还想屏蔽哪些信号，并在信号处理结束后，自动解除对它们的屏蔽
  - struct sigaction sigact = {};
    sigact.sa_handler = oldsigint;
    sigaddset (&sigact.sa_mask, SIGQUIT);
    if (sigaction (SIGINT, &sigact, NULL) == -1) {
        perror ("sigaction");
        exit (EXIT_FAILURE);
    }

知识讲解

增减信号掩码（续1）

Technology Tarena 达内科技

- 另一方面，还可以通过sigaction::sa_flags成员的SA_NODEFER或SA_NOMASK标志位，告诉系统内核在信号处理函数执行期间，不要屏蔽这个正在被处理的信号
  - struct sigaction sigact = {};
    sigact.sa_handler = oldsigint;
    sigaddset (&sigact.sa_mask, SIGQUIT);
    sigact.sa_flags = SA_NOMASK;
    if (sigaction (SIGINT, &sigact, NULL) == -1) {
        perror ("sigaction");
        exit (EXIT_FAILURE);
    }

知识讲解

提供更多信息的信号处理函数

Technology Tarena 达内科技

- 旧风格信号处理函数的原型类似下面这个样子
  void handler (int ***signum***);
  该函数只有唯一的参数，表示触发该函数被执行的信号
  sigaction::sa_handler成员所指向的就是这样的函数，
  默认情况下，sigaction函数和signal函数向系统内核注册的信号处理函数具有完全相同的接口形式
- 新风格信号处理函数的原型类似下面这个样子
  void action (int ***signum***, siginfo_t* ***siginf***,
  void* ***reserved***);
  - ***signum***：信号编号
  - ***siginf***：信号信息
  - ***reserved***：预留参数，目前不用

知识讲解

提供更多信息的信号处理函数（续1）

Technology Tarena 达内科技

- 为了使用新风格的信号处理函数，除了要让sigaction::sa_sigaction成员指向类似前面的action函数以外，还要给sigaction::sa_flags成员加上SA_SIGINFO标志位
  - struct sigaction sigact = {};
    sigact.sa_sigaction = newsigint;
    sigaddset (&sigact.sa_mask, SIGQUIT);
    sigact.sa_flags = SA_NOMASK | SA_SIGINFO;
    if (sigaction (SIGINT, &sigact, NULL) == -1) {
        perror ("sigaction");
        exit (EXIT_FAILURE);
    }

知识讲解

提供更多信息的信号处理函数（续2）

Technology Tarena 达内科技

- 相比于旧风格的信号处理函数，新风格的信号处理函数除提供信号编号外，还通过一个siginfo_t类型的结构体向用户提供了更多有关触发这次响应行为的信号的细节
  - typedef struct siginfo {
    // 发送信号进程的PID
    pid_t si_pid;
    // 信号附加数据
    sigval_t si_value;
    ...
    } siginfo_t;
  - siginfo_t结构体类型共包含18个成员，对信号提供了非常详尽的描述，但其中对用户最具价值只有上面列出的两个

知识讲解

提供更多信息的信号处理函数（续3）

Technology Tarena 达内科技

- 例如
  - void newsigint (int signum, siginfo_t* siginf,
    void* reserved) {
    printf ("%d进程给我发了一个SIGINT信号\n",
    siginf->si_pid);
    }
- 注意，siginfo_t结构中的另一个成员si_value，其实更有价值，通过它甚至可以向信号处理函数传递任意数量个类型任意的用户自定数据，但要想使用该成员必须通过sigqueue函数发送信号，传统的kill和raise函数没有给信号附加数据的能力

一次性信号处理

Technology Tarena 达内科技

- 如前文所述，在某些Unix系统上，通过signal函数注册的信号处理函数只能一次性有效。系统内核在每次调用信号处理函数之前，会先将对该信号的处理恢复为默认操作。为了获得持久有效的信号处理，人们不得不在每次处理完信号以后，再次调用signal函数重新注册一遍。这其实也是早期Unix系统信号机制可靠性差的一种表现
- 包括Linux在内的许多现代Unix系统，信号机制的可靠性已经获得极大提升，即使是通过传统意义上的signal函数注册的信号处理函数，也能做到持久有效
- 但也不排除仍旧存在着一些历史遗留的代码，为了达到某种特殊目的，恰恰利用了信号处理一次性有效的特性。为此，sigaction函数也提供了针对这种用法的兼容方案

一次性信号处理（续1）

Technology Tarena 达内科技

- 如果在sigaction::sa_flags成员中加上SA_ONESHOT或SA_RESETHAND标志位，那么每当执行完一次信号处理函数后，即将对该信号的处理恢复为默认操作。这就和老式signal函数，每次注册只管一次的行为方式一致了
  - struct sigaction sigact = {};
    sigact.sa_handler = oldsigint;
    sigaddset (&sigact.sa_mask, SIGQUIT);
    sigact.sa_flags = SA_NOMASK | SA_ONESHOT;
    if (sigaction (SIGINT, &sigact, NULL) == -1) {
        perror ("sigaction");
        exit (EXIT_FAILURE);
    }

知识讲解

Technology Tarena 达内科技

信号处理

【参见：sigact.c】

课堂练习

- 信号处理

自动重启被中断的系统调用

Technology Tarena 达内科技

- 缺省情况下，当进程正阻塞在某个系统中时，如果收到信号，系统将中断这个被阻塞的系统调用，转而执行相应的信号处理函数。待信号处理函数返回以后，之前被中断的系统调用将返回失败，同时置errno为EINTR，表示被信号中断。为了完成之前未完成的任务，往往需要做一些特殊处理
  - ssize_t len;
    char buf[256];
    again:
    if ((len = read (fd, buf, sizeof (buf))) == -1) {
    if (errno == EINTR) goto again;
    perror ("read"); exit (EXIT_FAILURE);
    }

知识讲解

自动重启被中断的系统调用（续1）

Technology Tarena 达内科技

- 如果在sigaction::sa_flags成员中加上SA_RESTART标志位，那么被信号中断的系统调用，将在相应的信号处理函数返回以后，自动恢复被中断前的操作
  - ssize_t len;
    char buf[256];
    if ((len = read (fd, buf, sizeof (buf))) == -1) {
        perror ("read");
        exit (EXIT_FAILURE);
    }

知识讲解

Technology Tarena 达内科技

自动重启

【参见：restart.c】

课堂练习

- 自动重启

Technology Tarena 达内科技

信号发送

sigqueue

Technology Tarena 达内科技

- 向指定进程发送附加了数据的信号

```
#include <signal.h>

int sigqueue (pid_t pid, int signum,
    const union sigval value);
```

成功返回0，失败返回-1

  - *pid*：接收信号进程的PID
  - *signum*：信号编号
  - *value*：附加数据
- 向*pid*进程发送*signum*信号，附加*value*数据

为信号提供附加数据

Technology Tarena 达内科技

- 为了在向特定进程发送信号的同时附加用户自己的数据，可以通过sigqueue函数的***value***参数达到目的。注意该参数的类型sigval是个联合，与siginfo_t::si_value成员的类型sigval_t其实是一样的
  - typedef union sigval {
    int sival_int;
    void* sival_ptr;
    } sigval_t;
  - typedef struct siginfo {
    pid_t si_pid;
    sigval_t si_value;
    ...
    } siginfo_t;

知识讲解

为信号提供附加数据（续1）

Technology Tarena 达内科技

- 如果伴随信号一起发送的附加数据用一个简单的整数就足以表达，那么使用sigval_t::sival_int成员无疑是最佳选择
- 但如果附加数据不只一个或者所包含的信息比较复杂无法用一个简单的整数来表示，那么不妨把它们放到一个结构体型的变量中，然后将该结构体变量的指针通过sigval_t::sival_ptr成员发送给信号处理函数
- 但是请注意，如果伴随信号的附加数据是一个指针，那么一定要保证该指针所指向的内存区域，在信号处理函数执行期间必须有效

知识讲解

为信号提供附加数据（续2）

Technology Tarena 达内科技

- 例如
  - STUDENT* student = malloc (sizeof (STUDENT));
    strcpy (student->name, "张飞");
    student->age = 25;
    sigval_t value;
    value.sival_ptr = student;
    if (sigqueue (pid, SIGINT, value) == -1) {
        perror ("sigqueue"); exit (EXIT_FAILURE); }
  - void sigint (int signum, siginfo_t* siginf, void* reserved) {
        STUDENT* student =
            (STUDENT*)siginf->si_value.sival_ptr;
        printf ("%s, %d\n", student->name, student->age);
        free (student);
    }

现代与古典

Technology Tarena 达内科技

- 无论是较现代的sigqueue函数，还是更古典的kill函数和raise函数，它们在发送信号时都不会等待信号处理的结束。也就是说这些函数的返回并不意味着所发送的信号已被处理完毕。它们只负责把信号交给内核，至于内核什么时候递送，递送不递送(比如被信号掩码屏蔽的情况)，信号处理函数有没有对这些信号做处理，处理的结果如何，成功了还是失败了，函数的调用者一概无从得知
- 与kill和raise函数不同的是sigqueue函数可以保证所发送信号的顺序性，即先发出的一定先收到，后发出的一定后收到，这就是所谓“信号队列”的语义，kill和raise保证不了这一点。但这仅限于可靠信号的情况，如果是不可靠信号，无论现代还是古典，都是一样地照丢不误

Technology Tarena 达内科技

信号发送

【参见：sigque.c、send.c、recv.c】

课堂练习

- 信号发送

定时器

Technology Tarena 达内科技

系统计时器

系统计时器

Technology Tarena 达内科技

- 运行一个进程所消耗的时间包括三个部分
  - 用户时间：进程消耗在用户态的时间
  - 内核时间：进程消耗在内核态的时间
  - 睡眠时间： 进程消耗在等待I/O、睡眠等不被调度的时间

- 系统内核为系统中的每个进程维护三个计时器
  - 真实计时器：统计进程的执行时间
  - 虚拟计时器：统计进程的用户时间
  - 实用计时器：统计进程的用户时间和内核时间之和

知识讲解

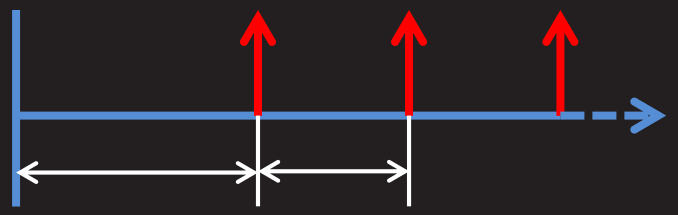
Technology Tarena 达内科技

设置定时器

设置定时器

Technology Tarena 达内科技

- 三个系统计时器除了统计进程的各种时间以外，还可以按照各自的计时规则，以定时器的方式工作，向进程周期性地发送不同的信号
  - SIGALRM (14)：真实定时器到期
  - SIGVTALRM (26)：虚拟定时器到期
  - SIGPROF (27)：实用定时器到期
- 定时器在可以发送信号之前，必须先行设置。每个定时器均包括两个属性，需要在设置时初始化好
  - 初始间隔：从设置定时器到它首次发出信号的时间间隔
  - 重复间隔：定时器发出的两个相邻信号之间的时间间隔



设置定时器（续1）

Technology Tarena 达内科技

- 设置、启动、关闭定时器

```
#include <sys/time.h>

int setitimer (int which,
    const struct itimerval* new_value,
    struct itimerval* old_value);
```

成功返回0，失败返回-1

- *which*：指定哪个定时器，可取以下值
  - `ITIMER_REAL` - 真实定时器
  - `ITIMER_VIRTUAL` - 虚拟定时器
  - `ITIMER_PROF` - 实用定时器
- *new_value*：新设置值

设置定时器（续2）

Technology Tarena 达内科技

- 设置、启动、关闭定时器
  - ***old_value***：输出原设置值，可置NULL
- 该函数有关定时器设置的参数都选用了itimerval类型
  - struct itimerval {

```
// 重复间隔，取0将使定时器在发送第一个信号后停止
struct timeval it_interval;
// 初始间隔，取0将立即停止定时器
struct timeval it_value;
};
```

  - struct timeval {

```
long tv_sec;  // 秒数
long tv_usec; // 微秒数
};
```

知识讲解

设置定时器（续3）

Technology Tarena 达内科技

- 例如
  - 5秒以后开始计时，每3毫秒计时一次

```
struct itimerval it;
it.it_value.tv_sec = 5;
it.it_value.tv_usec = 0;
it.it_interval.tv_sec = 0;
it.it_interval. tv_usec = 3000;
if (setitimer (ITIMER_REAL, &it, NULL) == -1) {
    perror ("setitimer");
    exit (EXIT_FAILURE); }
```

  - 关闭定时器

```
it.it_value.tv_sec = 0;
setitimer (ITIMER_REAL, &it, NULL);
```


Technology Tarena 达内科技

电子秒表

【参见：timer.c】

课堂练习

- 电子秒表

获取定时器设置

Technology Tarena 达内科技

- 获取定时器设置

```
#include <sys/time.h>

int getitimer (int which, struct itimerval* curr_value);
```

成功返回0，失败返回-1

  - *which*：指定哪个定时器，可取以下值

    `ITIMER_REAL` - 真实定时器

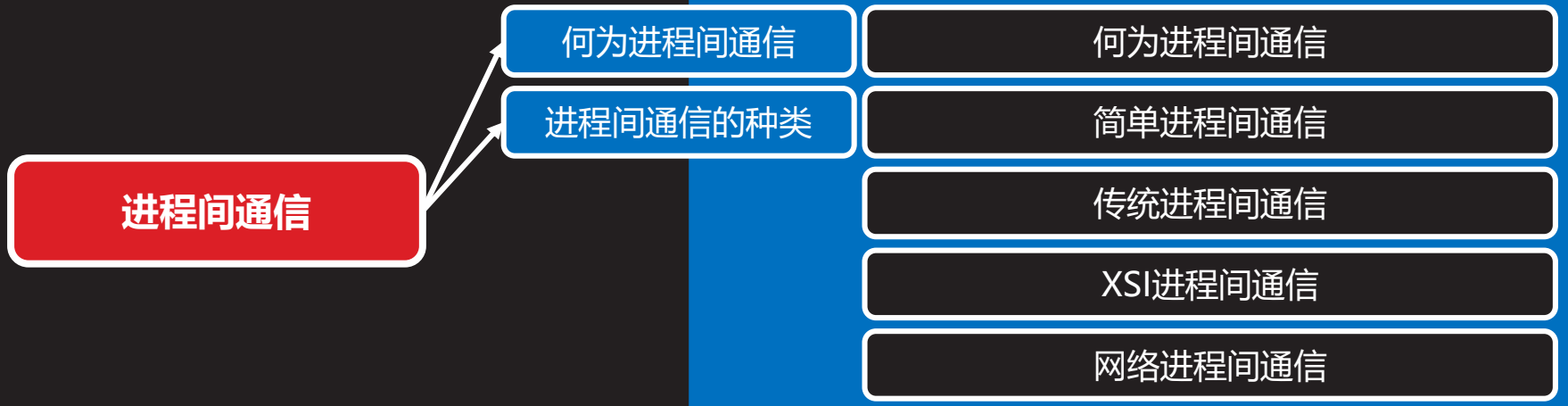
    `ITIMER_VIRTUAL` - 虚拟定时器

    `ITIMER_PROF` - 实用定时器

  - *curr_value*：输出当前设置值

知识讲解

进程间通信


进程间通信
何为进程间通信
进程间通信的种类
何为进程间通信
简单进程间通信
传统进程间通信
XSI进程间通信
网络进程间通信

Technology Tarena 达内科技

何为进程间通信

何为进程间通信

Technology Tarena 达内科技

- 正如前文所述，Unix/Linux系统中的每个进程都拥有4G字节大小专属于自己的虚拟内存空间
- 除去内核空间的1G以外，每个进程都有一张独立的内存映射表(又名内存分页表)记录着虚拟内存页和物理内存页之间的映射关系
- 同一个虚拟内存地址，在不同的进程中，会被映射到完全不同的物理内存区域，因此在多个进程之间以交换虚拟内存地址的方式交换数据是不可能的
- 鉴于进程之间天然存在的内存壁垒，要想实现多个进程间的数据交换，就必须提供一种专门的机制，这就是所谓的进程间通信(InterProcess Communication, IPC)

Technology Tarena 达内科技

进程间通信的种类

简单进程间通信

Technology Tarena 达内科技

- 命令行参数
  - 在通过exec函数创建新进程时，可以为其指定命令行参数，借助命令行参数可以将创建者进程的某些数据传入新进程

```
execl ("login", "login", username, password, NULL);
```

- 环境变量
  - 类似地，也可以在调用exec函数时为新进程提供环境变量

```
sprintf (envp[0], "USERNAME=%s", username);
sprintf (envp[1], "PASSWORD=%s", password);
execle ("login", "login", NULL, envp);
```

- 内存映射文件
  - 通信双方分别将自己的一段虚拟内存映射到同一个文件中
- 信号
  - SIGUSR1/SIGUSR2，sigaction/sigqueue/附加数据

知识讲解

传统进程间通信

Technology Tarena 达内科技

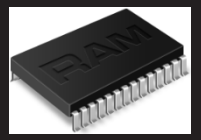
- 管道
  - 管道是Unix系统中最古老的进程间通信方式，并且所有的Unix系统和包括Linux系统在内的各种类Unix系统也都提供这种进程间通信机制。管道有两种限制
    - 管道都是半双工的，数据只能沿着一个方向流动
    - 管道只能在具有公共祖先的进程之间使用。通常一个管道由一个进程创建，然后该进程通过fork函数创建子进程，父子进程之间通过管道交换数据
  - 大多数Unix/Linux系统除了提供传统意义上的无名管道以外，还提供有名管道，对后者而言第二种限制已不复存在
  - 基于SVR4的管道事实上是全双工的，即数据可在一根管道中双向流动。但是POSIX.1只提供半双工管道，出于移植性的考虑，不妨依然坚持“管道都是半双工的”这一假定

知识讲解

XSI进程间通信

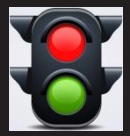
Technology Tarena 达内科技

- 共享内存
  - 共享内存允许两个或两个以上的进程共享同一块给定的内存区域。因为数据不需要在通信诸方之间来回复制，所以这是速度最快的一种进程间通信方式



- 消息队列
  - 消息队列是由系统内核负责维护并可在多个进程之间共享存取的消息链表。它的优点是：传输可靠、流量受控、面向有结构的记录、支持按类型过滤

- 信号量
  - 与共享内存和消息队列不同，信号量并不是为了解决进程间的数据交换问题。它所关注的是有限的资源如何在无限的用户间合理分配，即资源竞争问题



知识讲解

网络进程间通信

Technology Tarena 达内科技

- 套接字
  - 脱胎于System V的共享内存、消息队列和信号量，它们最大的问题就是都没有使用文件描述符。因此也就无法对它们使用多路I/O复用——select和poll。更不能象访问文件那样，访问系统内核中的IPC对象。从本质上讲这是对Unix系统向来秉承的一切皆文件思想的一种无耻的背叛
  - 从这个意义上讲，源自BSD的网络编程接口——套接字——恰恰体现了一种对Unix固有文化的虔诚的皈依，因为它完全是基于文件描述符的。所有对文件起作用的系统调用，无论是基本的读写、还是对内核对象的控制，甚或是多路I/O复用，都可以无一例外地应用于网络或本机中的不同进程。这足以证明BSD把一切皆文件进行到底的决心和斗志

知识讲解

管道

Technology Tarena 达内科技

有名管道

有名管道

Technology Tarena 达内科技

- 有名管道亦称FIFO，是一种特殊的文件，它的路径名存在于文件系统中。通过mkfifo命令可以创建管道文件

```
$ mkfifo myfifo
```

- 在文件系统中，管道文件被显示成这个样子

```
prw-rw-r-- 1 tarena tarena 0 12月  5 10:07 myfifo
```

- 即使是毫无亲缘关系的进程，也可以通过管道文件通信

```
$ echo 'Hello, FIFO !' > myfifo
```

```
$ cat myfifo
```

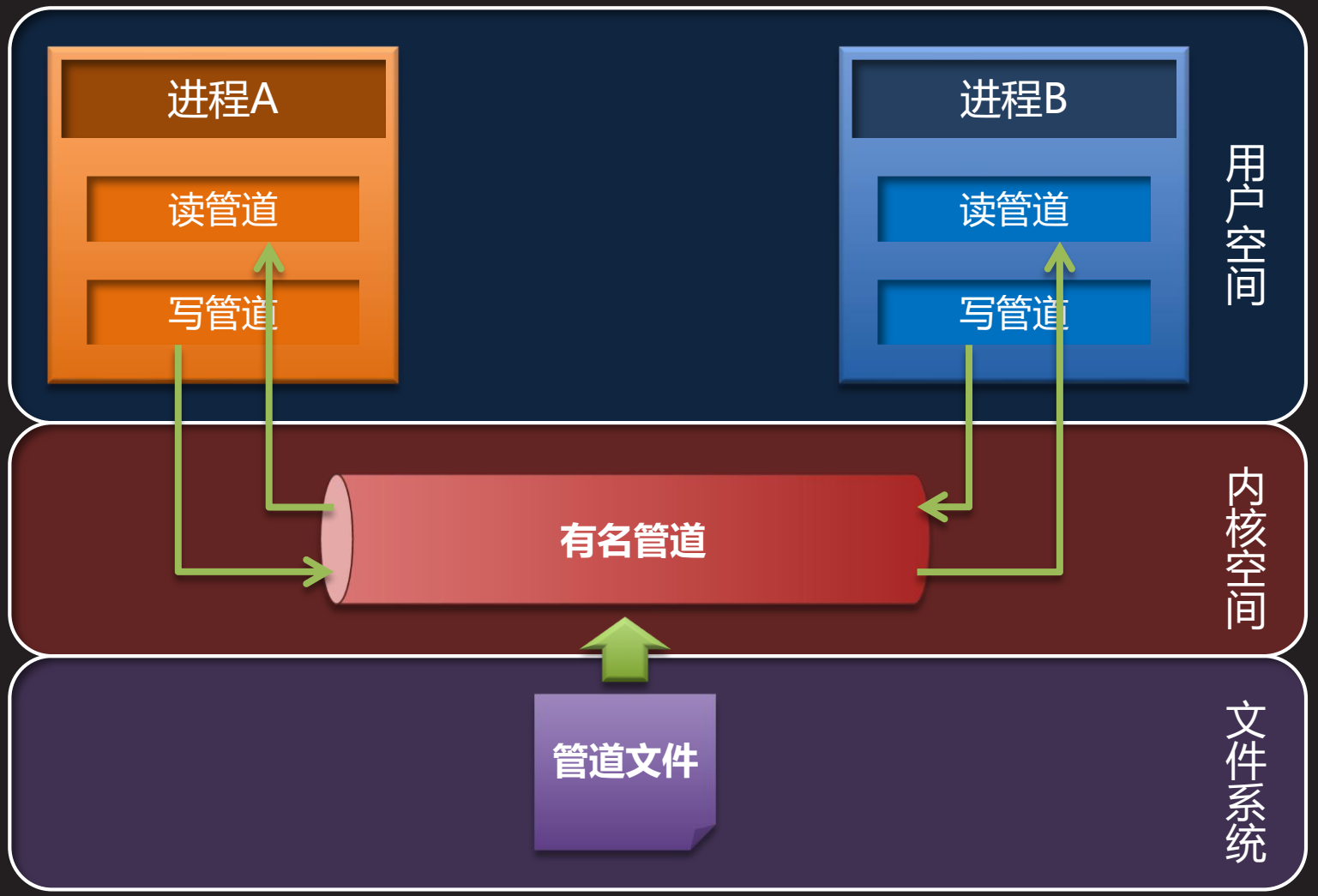
```
Hello, FIFO !
```

- 管道文件在磁盘上只有i节点没有数据块，也不保存数据

知识讲解

有名管道（续1）

- 基于有名管道实现进程间通信的逻辑模型



知识讲解

有名管道（续2）

Technology Tarena 达内科技

- 有名管道不仅可以用于Shell命令，也可以在代码中使用
- 基于有名管道实现进程间通信的编程模型

步骤	进程A	函数	进程B	步骤
1	创建管道	mkfifo	——	——
2	打开管道	open	打开管道	1
3	读写管道	read/write	读写管道	2
4	关闭管道	close	关闭管道	3
5	删除管道	unlink	——	——

- 其中除了mkfifo函数是专门针对有名管道的，其它函数都与操作普通文件没有任何差别
- 有名管道是文件系统的一部分，如不删除，将一直存在

知识讲解

有名管道（续3）

Technology Tarena 达内科技

- 创建有名管道文件

```
#include <sys/stat.h>

int mkfifo (const char* pathname, mode_t mode);
```

成功返回0，失败返回-1

  - pathname：文件路径
  - mode：权限模式
- 例如
  -
```
if (mkfifo ("myfifo", 0666) == -1) {
    perror ("mkfifo");
    exit (EXIT_FAILURE);
}
```

知识讲解

Technology Tarena 达内科技

基于有名管道的进程间通信

【参见：wfifo.c、rfifo.c】

课堂练习

- 基于有名管道的进程间通信

Technology Tarena 达内科技

无名管道

无名管道

Technology Tarena 达内科技

- 无名管道是一个与文件系统无关的内核对象，主要用于父子进程之间的通信，需要用专门的系统调用函数创建

```
#include <unistd.h>

int pipe (int pipefd[2]);
```

成功返回0，失败返回-1

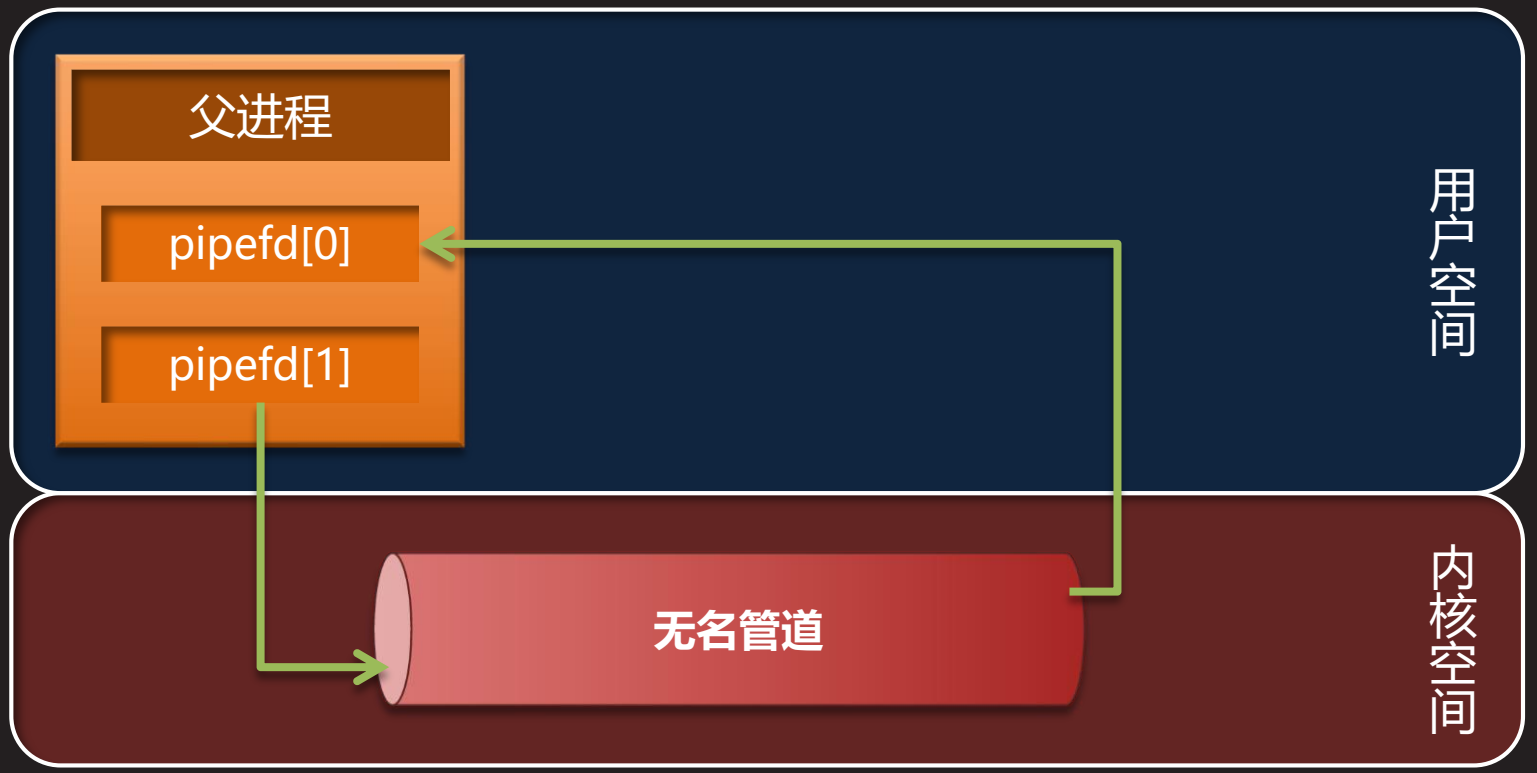
  - ***pipefd***：输出两个文件描述符，***pipefd***[0]用于从所创建的无名管道中读取数据，***pipefd***[1]用于向该管道写入数据

- 例如
  - int pipefd[2];
    if (pipe (pipefd) == -1) {
        perror ("pipe"); exit (EXIT_FAILURE); }

无名管道（续1）

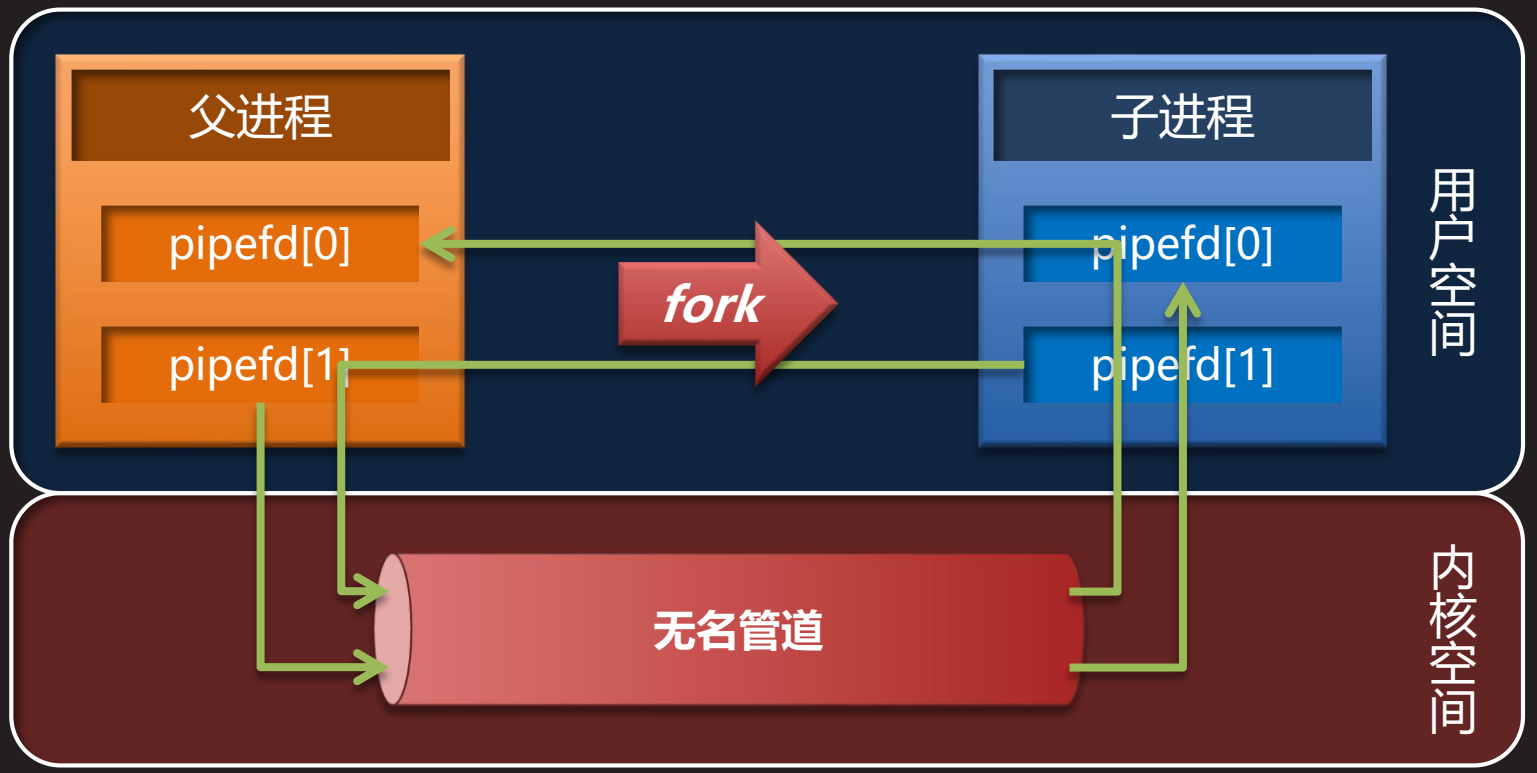
Technology Tarena 达内科技

- 基于无名管道实现进程间通信的编程模型
  1. 父进程调用pipe函数在系统内核中创建无名管道对象，并通过该函数的输出参数***pipefd***，获得分别用于读写该管道的两个文件描述符***pipefd***[0]和***pipefd***[1]



无名管道（续2）

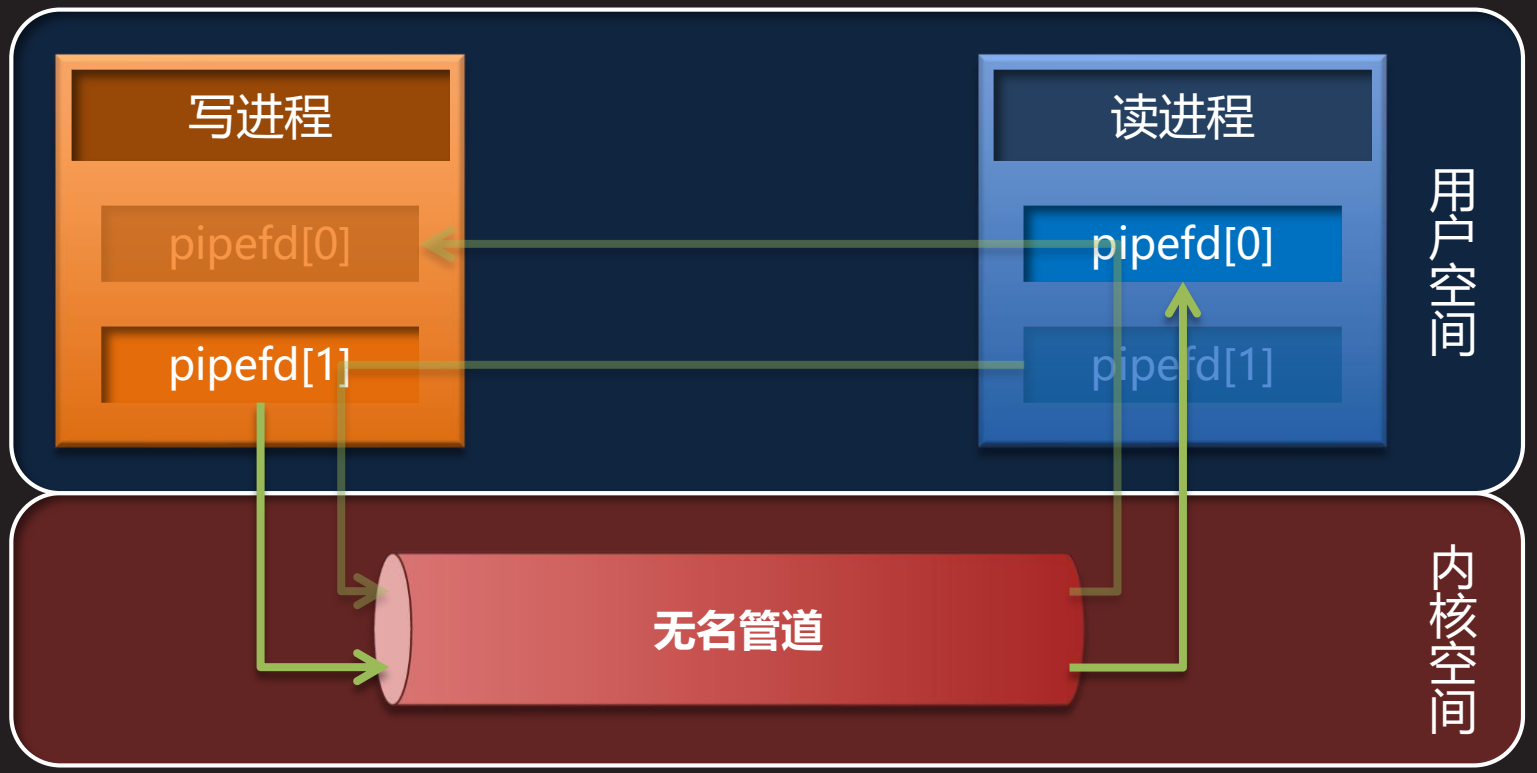
- 基于无名管道实现进程间通信的编程模型
  2. 父进程调用fork函数，创建子进程。子进程复制父进程的文件描述符表，因此子进程同样持有分别用于读写该管道的两个文件描述符***pipefd***[0]和***pipefd***[1]



知识讲解

无名管道（续3）

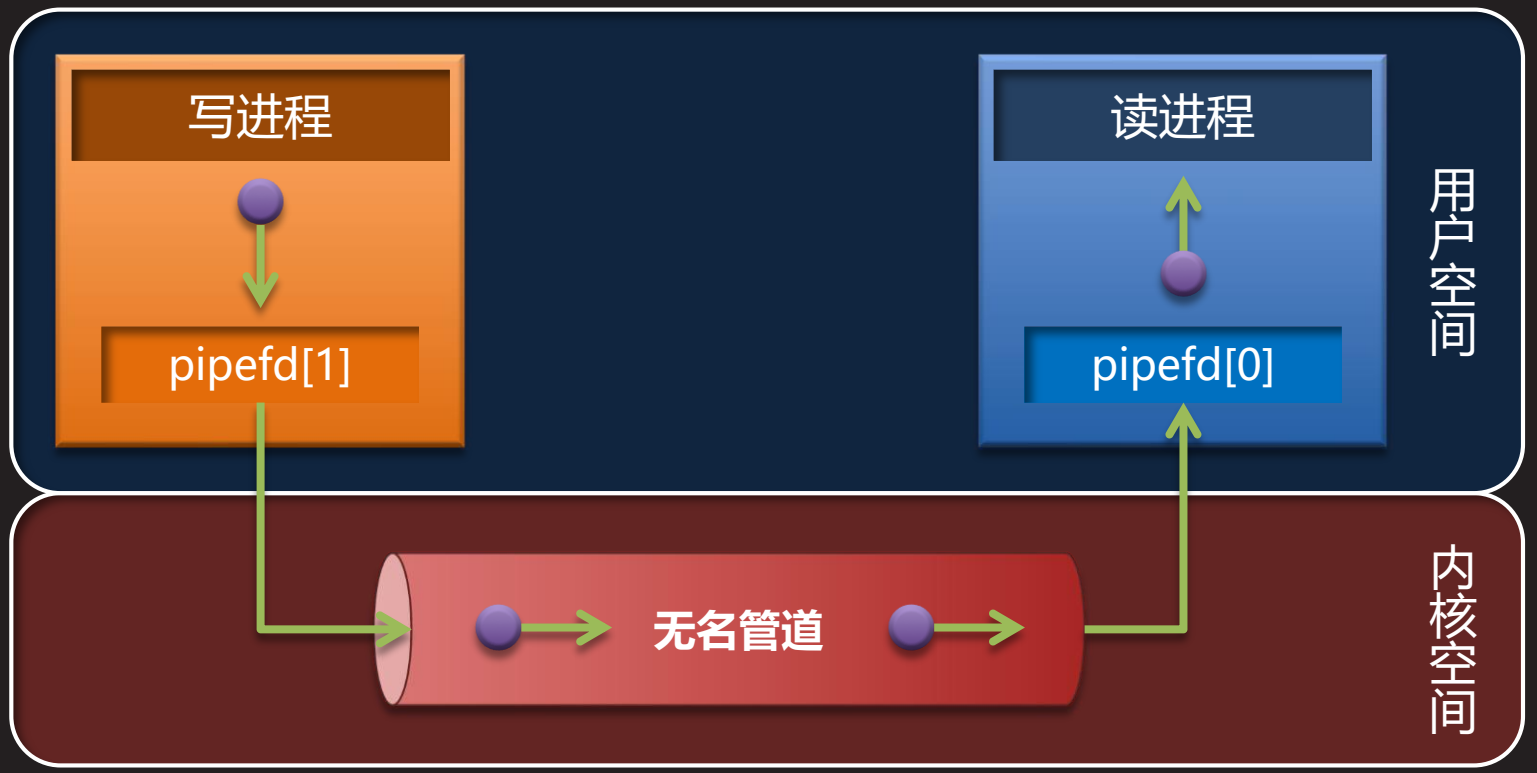
- 基于无名管道实现进程间通信的编程模型
  3. 负责写数据的进程关闭无名管道对象的读端文件描述符***pipefd***[0]，而负责读数据的进程则关闭该管道的写端文件描述符***pipefd***[1]



知识讲解

无名管道（续4）

- 基于无名管道实现进程间通信的编程模型
  4. 父子进程通过无名管道对象以半双工的方式传输数据。如果需要在父子进程间实现双向通信，较一般化的做法是创建两个管道，一个从父流向子，一个从子流向父



知识讲解

无名管道（续5）

- 基于无名管道实现进程间通信的编程模型
  5. 父子进程分别关闭自己所持有的写端或读端文件描述符。在与一个无名管道对象相关联的所有文件描述符都被关闭以后，该无名管道对象即从系统内核中被销毁

知识讲解

基于无名管道的进程间通信

【参见：pipe.c】

课堂练习

- 基于无名管道的进程间通信

Technology Tarena 达内科技

特殊情况

特殊情况

Technology Tarena 达内科技

- 无论是有名管道，还是无名管道，在对它们进行读写操作的过程中，都难免会遇到一些特殊情况
- 从写端已关闭的管道读取
  - 只要管道中还有数据，依然可以正常读取，一直读到管道中没有数据了，这时read函数会返回0(既不是返回-1，也不是阻塞)指示读到文件尾
- 向读端已关闭的管道写入
  - 会直接触发SIGPIPE(13)信号。该信号的默认操作是终止执行写入动作的进程。但如果执行写入动作的进程事先已将对SIGPIPE(13)信号的处理设置为忽略或者捕获并从对该信号的处理函数中返回，则write函数会返回-1，并置errno为EPIPE

特殊情况（续1）

Technology Tarena 达内科技

- 系统内核会为每个管道维护一个内存缓冲区，缓冲区的大小通常为4096字节，在<limits.h>头文件中被定义为PIPE_BUF宏
  - 如果写管道时发现缓冲区中的空闲空间不足以容纳此次write所要写入的字节，则write函数会阻塞，直到缓冲区中的空闲空间变得足够大为止
  - 如果同时有多个进程向同一个管道写入数据，而每次调用write函数写入的字节数都不大于PIPE_BUF，则这些write操作不会相互穿插，反之，如果单次写入的字节数了超过PIPE_BUF，则它们的write操作可能会相互穿插
  - 读取一个缓冲区为空的管道，将直接导致read函数阻塞
  - 若管道是非阻塞的，则上述所有的阻塞都会立即返回失败

知识讲解

Technology Tarena 达内科技

管道符号

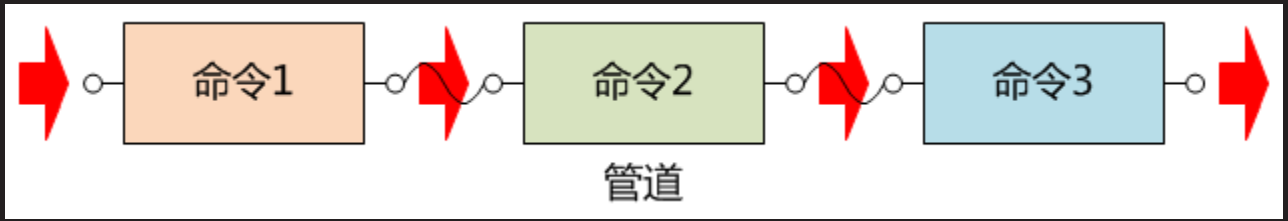
管道符号

- Unix/Linux系统中的多数Shell环境都支持通过管道符号“|”将前一个命令的输出作为后一个命令的输入的用法

```
$ ls -l /etc | more
```

```
总用量 1260
-rw-r--r--  1 root   root     2076  4月  3  2012 bash.bashrc
-rw-r--r--  1 root   root    58753  3月 31  2012 bash_completion
-rw-r--r--  1 root   root      356  4月 20  2012 bindresvport.blacklist
lrwxrwxrwx  1 root   root       15  6月 18 09:21 blkid.tab -> /dev/.blkid.tab
...
--更多--
```

- 系统管理员经常使用这种方法，把多个简单的命令连接成一条工具链，去解决一些通常看来可能很复杂的问题



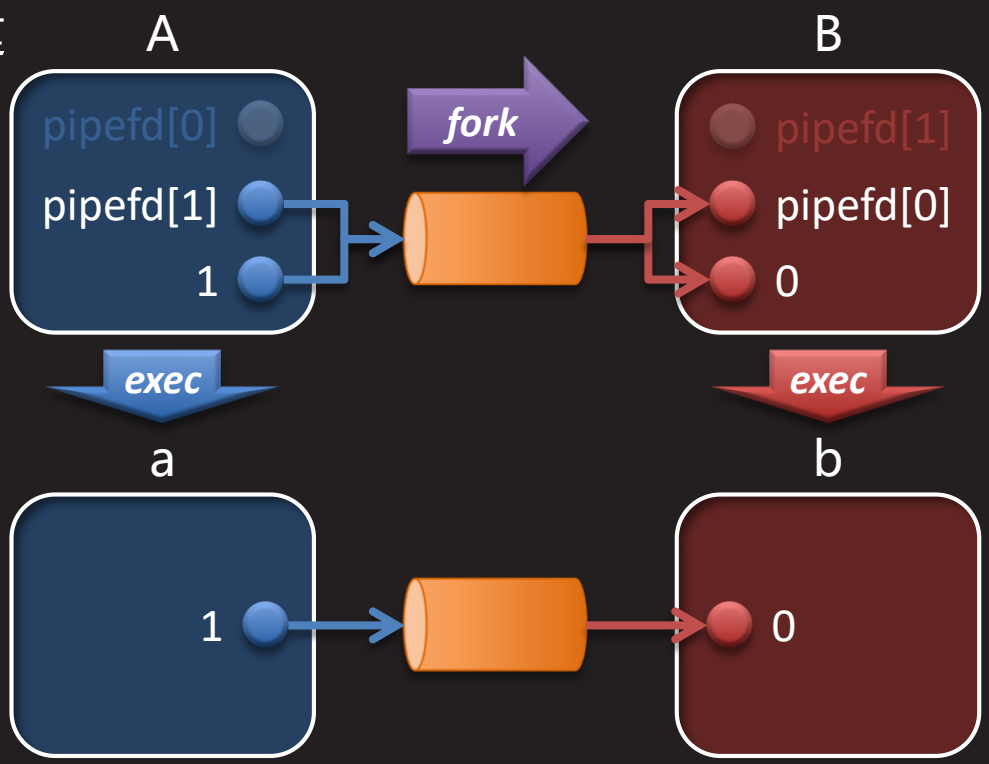
知识讲解

管道符号（续1）

Technology Tarena 达内科技

知识讲解

- 假设用户输入如下命令：a | b
  - Shell进程调用fork函数创建子进程A
  - 子进程A调用pipe函数创建无名管道，而后执行
    dup2 (pipefd[1],
    STDOUT_FILENO);
  - 子进程A调用fork函数创建子进程B，子进程B执行
    dup2 (pipefd[0],
    STDIN_FILENO);
  - 子进程A和子进程B分别调用exec函数创建a、b进程



  - a进程所有的输出都通过写端进入管道，而b进程所有的输入则统统来自该管道的读端。这就是管道符号的工作原理

Technology Tarena 达内科技

模拟管道符号的实现

【参见：output.c、input.c、shell.c】

课堂练习

- 模拟管道符号的实现

Technology Tarena 达内科技

总结和答疑