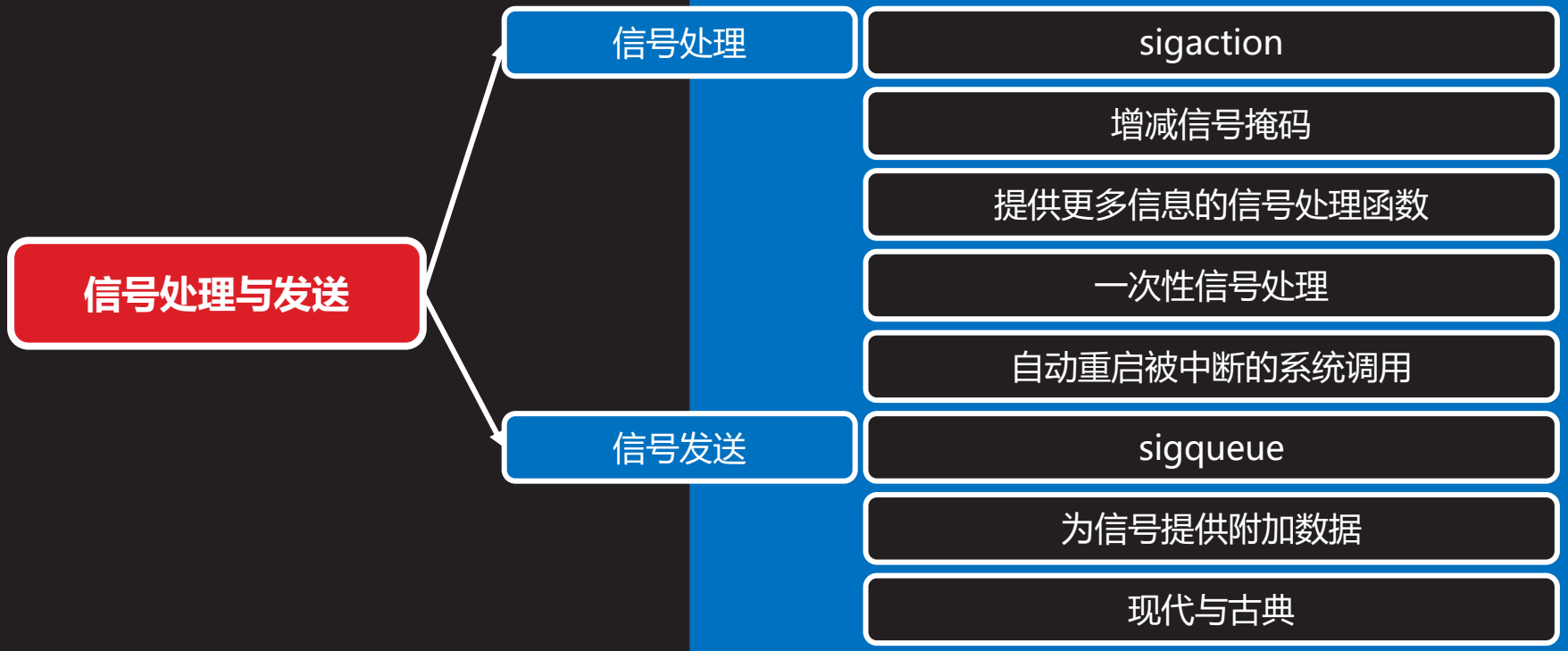
Technology Tarena 达内科技

Unix系统高级编程

现代风格的信号处理 Unit20

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

sigaction（续1）

- 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

- 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）

- 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

- 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）

- 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）

- 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）

- 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

- 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）

- 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

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

- 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）

- 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

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

- 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

知识讲解

为信号提供附加数据

- 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

信号发送

【参见：sigque.c】

课堂练习

- 信号发送

为信号提供附加数据（续1）

- 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）

- 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);
    }

知识讲解

基于信号的进程间通信

【参见：send.c、recv.c】

课堂练习

- 基于信号的进程间通信

现代与古典

- 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

总结和答疑