信号
Signal

课 程 名 ： 计算机系统

主 讲 人 ： 孟文龙

信号

- 信号*signal*就是一条小消息，用以通知进程：系统中发生了一个某种类型的事件
  - 类似于异常和中断
  - 由内核发送至进程（有时是在另一个进程的请求下）
  - 信号类型是用一个较小的整数 ID 来标识的（1-30, 传统 UNIX）/ 现代 64 类信号
  - 信号仅传递两样信息：信号的 ID 和信号到达的事实

ID	名称	默认行为	相应事件
2	SIGINT	终止	来自键盘的中断（Ctrl-C）
9	SIGKILL	终止	杀死程序（该信号不能被捕获不能被忽略）
11	SIGSEGV	终止并转储	内存访问越界（段故障）
14	SIGALRM	终止	来自 alarm 函数的定时器信号
17	SIGCHLD	忽略	通知父进程它的一个子进程停止或者终止

Linux 异常对应的信号名和处理程序名

异常类型号	助记符	含义描述	处理程序名	信号名
0	#DE	除法出错	divide_error()	SIGFPE
1	#DB	单步跟踪	debug()	SIGTRAP
2		NMI 中断	nmi()	无
3	#BP	断点	int3()	SIGTRAP
4	#OF	溢出	overflow()	SIGFPE
5	#BR	边界检测（BOUND）	bounds()	SIGSEGV
6	#UD	无效操作码	invalid()	SIGILL
7	#NM	协处理器不存在	device_not_available()	无
8	#DF	双重故障	doublefault()	无
9	#MF	协处理器段越界	coprocessor_segment_overrun()	SIGFPE
10	#TS	无效 TSS	Invalid_tss()	SIGSEGV
11	#NP	段不存在	Segment_not_present()	SIGBUS
12	#SS	栈段错	Stack_segment()	SIGBUS
13	#GP	一般性保护错（GPF）	General_protection()	SIGSEGV
14	#PF	页故障	Page_fault()	SIGSEGV
15		保留	无	无
16	#MF	浮点错误	Coprocessor_error()	SIGFPE
17	#AC	对齐检测	Alignment_check()	SIGSEGV
18	#MC	机器检测异常	Machine_check()	无
19	#XM	SIMD 浮点异常	Simd_coprocessor_error()	SIGFPE

信号术语：信号的发送相关

- 内核通过更新目的进程的某个状态，将一个信号发送给目标进程

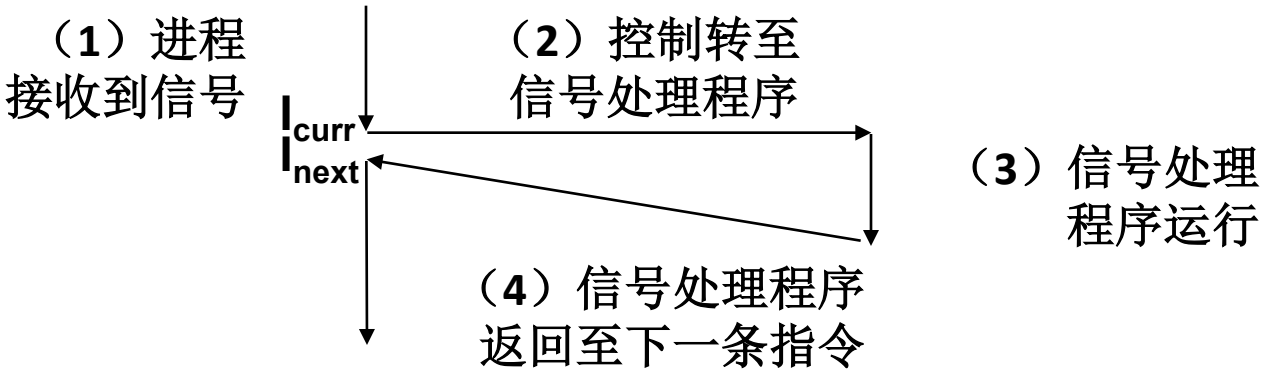
- 信号的发送可以出于如下 2 个原因之一：
  - 内核检测到一个系统事件，如除零错误（SIGFPE 信号）或子进程终止（SIGCHLD 信号）
  - 另一个进程使用了 kill() 系统调用，显式地请求内核发送一个信号至目的进程
    - 进程也可以给自己发送信号

c

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

int main() {
    int *p = NULL;
    if (!p) {
        printf("Null pointer detected, exiting...\n");
        kill(getpid(), SIGKILL); // 主动给自己发SIGKILL信号
    }
    printf("This line will not be executed\n");
    return 0;
}
```

信号术语：信号的接收

- 当目的进程在内核的强制下，以某种方式对信号的发送做出某种反应时，它就接收了信号
- 信号的接收有如下几种可能：
  - 忽略(*ignore*)该信号（什么也不做,适用于一些不重要或无需处理的信号）
  - 进程终止（可能会转储一些信息便于调试）或停止
  - 捕获该信号（通过执行一个称作信号处理程序的软件异常处理程序）



信号术语：待处理信号和阻塞信号

- 一个已经发出但没有被接收的信号叫做待处理信号 pending signal
  - 任何时刻，一种类型最多只有一个待处理信号
    - 如果一个进程有一个类型为 *k* 的待处理信号，那么此后任何发送到这个进程的类型为 *k* 的信号都会被丢弃

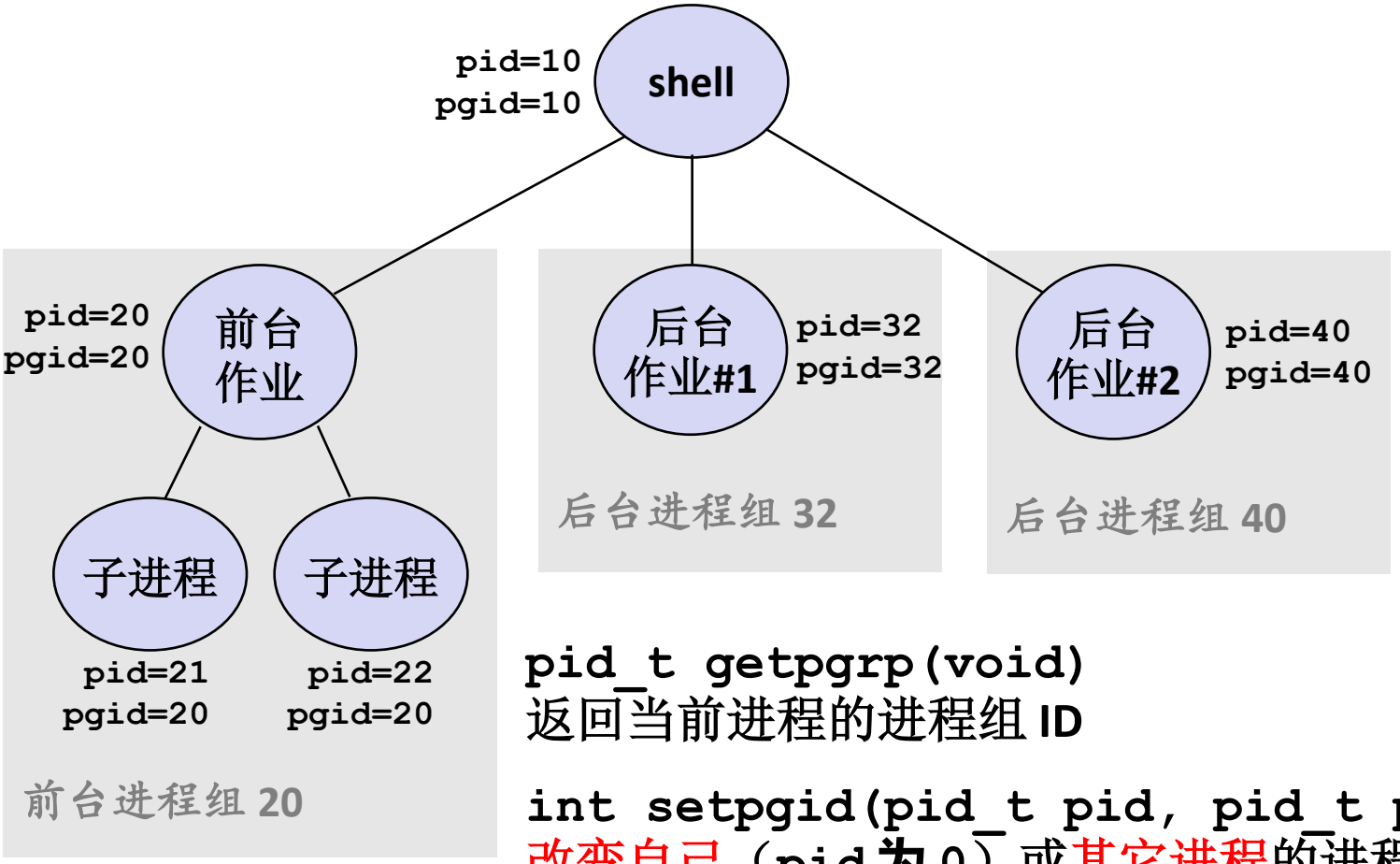
- 一个进程也可以有选择地阻塞对某种信号的接收
  - 阻塞的信号仍然可以发送，但不会被接收，直到进程取消对该信号的阻塞

信号术语：待处理位与阻塞位

- 内核在每个进程中维护着**待处理位向量**`pending`和**阻塞位向量** `blocked`
  - `pending` 表示待处理信号的集合
    - 若发送了一个类型为 *k* 的信号，内核会将 `pending` 中的第 *k* 位置 1
    - 若接收了一个类型为 *k* 的信号，内核会将 `pending` 中的第 *k* 位清 0
  - `blocked` 表示被阻塞信号的集合
    - 也称信号掩码signal mask或信号屏蔽字
    - 通过 `sigprocmask()` 函数设置和清除

信号的发送：进程组

- 为了方便信号管理和作业控制，把一组进程组织在一起。每个进程属于且仅属于一个进程组




```
pid_t getpgrp(void)
```
返回当前进程的进程组 ID

```
int setpgid(pid_t pid, pid_t pgid)
```
改变自己（pid为 0）或其它进程的进程组 ID

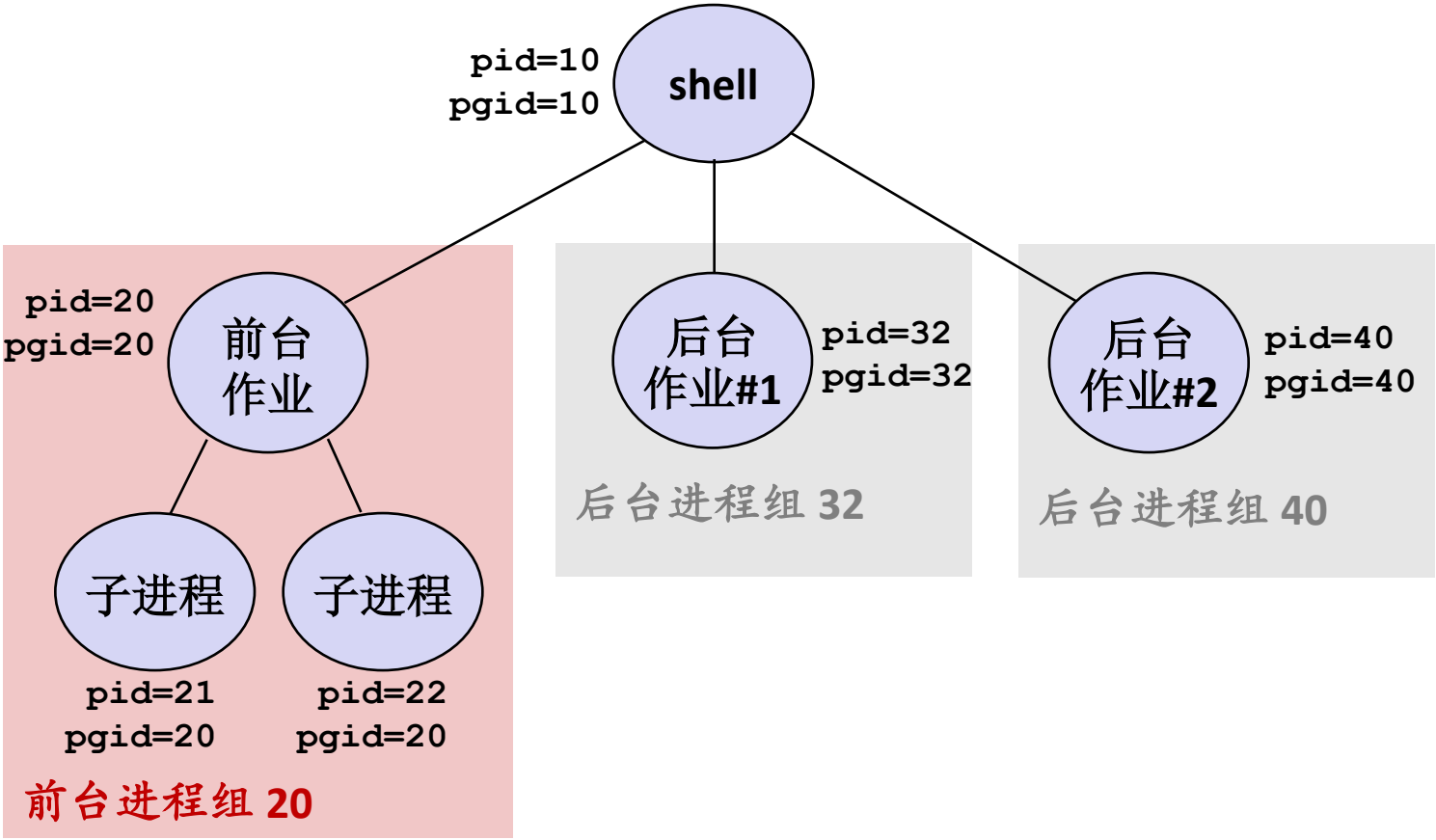
用 /bin/kill 程序发送信号

- /bin/kill 程序可以向某个进程或进程组发送任意信号
- 举例
  - /bin/kill -9 24818
    发送信号 9（SIGKILL）给进程 24818
  - /bin/kill -9 -24817
    发送信号 SIGKILL 给进程组 ID 为 24817 的所有进程
    - 若 PID 前加负号，则表示将信号发送至进程组 ID 等于 PID 的每一个进程
- 默认发送的信号是 15 号（SIGTERM，终止）

```
linux> ./forks
Child1: pid=24818 pgrp=24817
Child2: pid=24819 pgrp=24817

linux> ps
  PID TTY          TIME CMD
24788 pts/2    00:00:00 tcsh
24818 pts/2    00:00:02 forks
24819 pts/2    00:00:02 forks
24820 pts/2    00:00:00 ps
linux> /bin/kill -9 -24817
linux> ps
  PID TTY          TIME CMD
24788 pts/2    00:00:00 tcsh
24820 pts/2    00:00:00 ps
linux>
```

从键盘发送信号

- 键入Ctrl-C（Ctrl-Z）会导致内核发送一个 SIGINT（SIGTSTP）信号到前台作业对应的进程组中的每一个进程
  - SIGINT：默认情况是终止前台作业
  - SIGTSTP：默认情况是停止（挂起）前台作业



用 kill 函数发送信号

```
int kill(pid_t pid,
         int sig);
```

- ✓ 若 pid > 0，则将 sig 信号发送至进程 pid
- ✓ 若 pid 为 0（-1），则将 sig 发至本进程组中（本进程有权发给）的每个进程，包括自身
- ✓ 若 pid < -1，则将 sig 发送至进程组 -pid 中的所有进程

```
void fork12()
{
    pid_t pid[N];
    int i;
    int child_status;

    for (i = 0; i < N; i++)
        if ((pid[i] = Fork()) == 0) {
            /* 子进程无限循环 */
            while(1);
        }

    for (i = 0; i < N; i++) {
        printf("Killing process %d\n", pid[i]);
        kill(pid[i], SIGINT);
    }

    for (i = 0; i < N; i++) {
        pid_t wpid = wait(&child_status);
        if (WIFEXITED(child_status))
            printf("Child %d terminated with exit status %d.\n",
                   wpid, WEXITSTATUS(child_status));
        else
            printf("Child %d terminated abnormally.\n", wpid);
    }
}
```

forks.c

信号的接收

- 假设内核正在从异常处理程序返回，并准备将控权交给进程 p
  - 内核计算 `pnb = pending & ~blocked`
    - 即进程 p 的未被阻塞的待处理信号的集合
  - 若 pnb 为 0
    - 则将控制传递到 p 的逻辑控制流中的下一条指令
  - 否则
    - 选择 pnb 的最低非零位 *k*，强制 p 接收信号 *k*（清零）
    - 对信号的接收会触发进程 p 采取某种行为
    - 对 pnb 中所有的非零位 *k* 重复上述过程，直到所有未被阻塞的待处理信号都被处理完毕。
    - 控制传递到 p 的逻辑控制流中的下一条指令

默认行为

- **每个信号类型都有一个预定义的默认行为*default action*，包括：**
  - **接收进程终止**(SIGTERM、SIGINT)
  - **接收进程终止并转储**, 用于调试分析。(SIGSEGV（段错误）、SIGABRT（异常终止）等信号。)
  - **接收进程停止（挂起），直到被 `SIGCONT` 信号重启**(SIGSTOP、SIGTSTP 等信号)
  - **接收进程将该信号忽略**(SIGCHLD)

设置信号处理程序

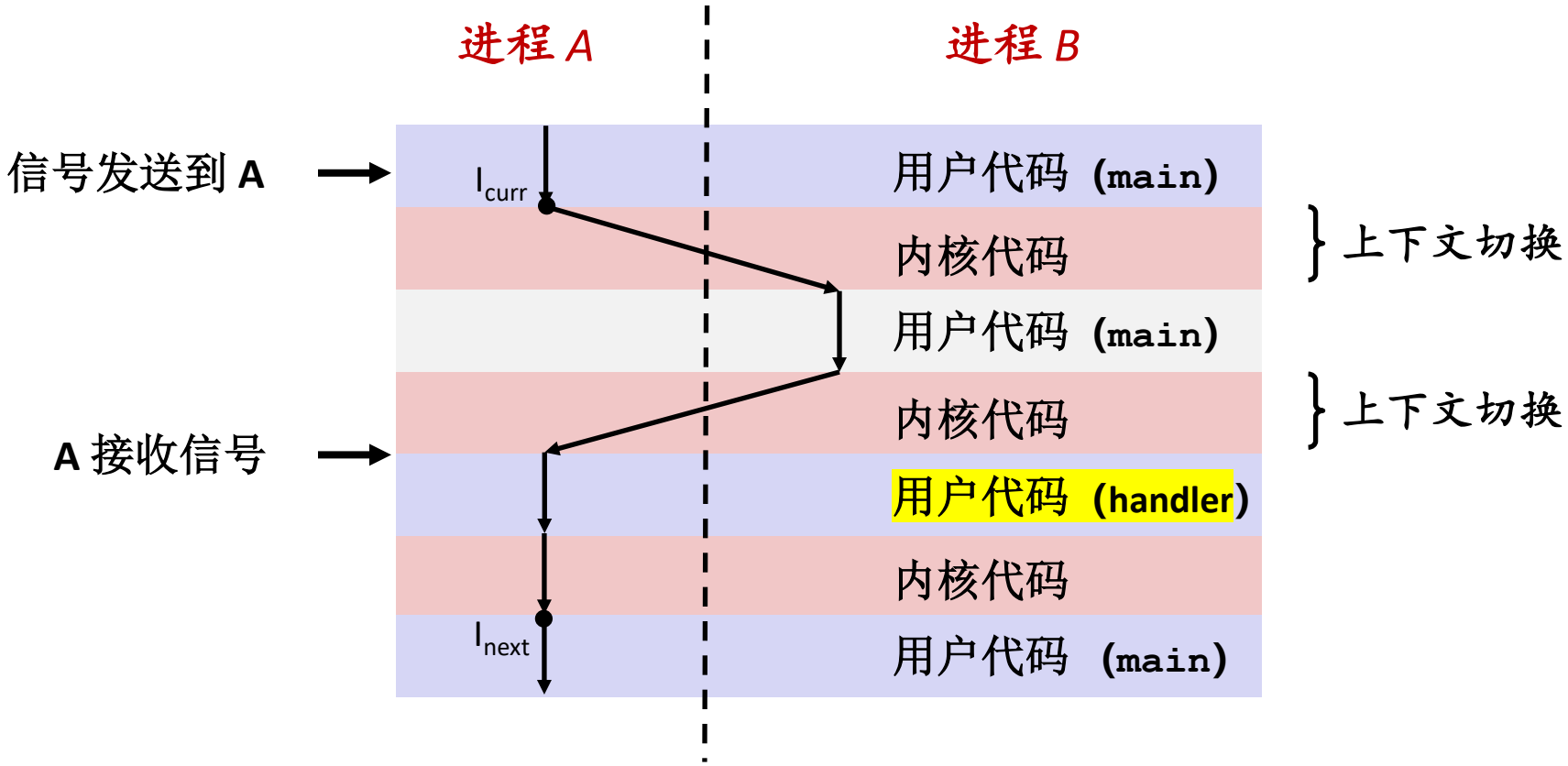
sighandler_t 定义见 8.5.3

- 可使用 signal 函数修改信号 signum 的默认行为
  - `sighandler_t signal(int signum,`
    `sighandler_t handler);`
- handler 的取值
  - SIG_IGN：忽略类型为 signum 的信号，即进程收到该信号时什么都不做。
  - SIG_DFL：将类型为 signum 的信号接收方式恢复至默认行为
  - 否则，handler 就是一个用户级函数（称作信号处理程序*signal handler*）的地址
    - 进程接收类型为 signum 的信号即调用信号处理程序
    - 将信号处理程序的入口地址传递到 signal 函数从而改变接收信号的默认行为，称作设置*install*信号处理程序
    - 信号处理程序的执行称作信号的捕获*catch*或处理*handle*
    - 当信号处理程序执行 return 时，控制会传递到控制流中被信号接收所中断的指令处

用信号处理程序捕获 SIGINT 信号

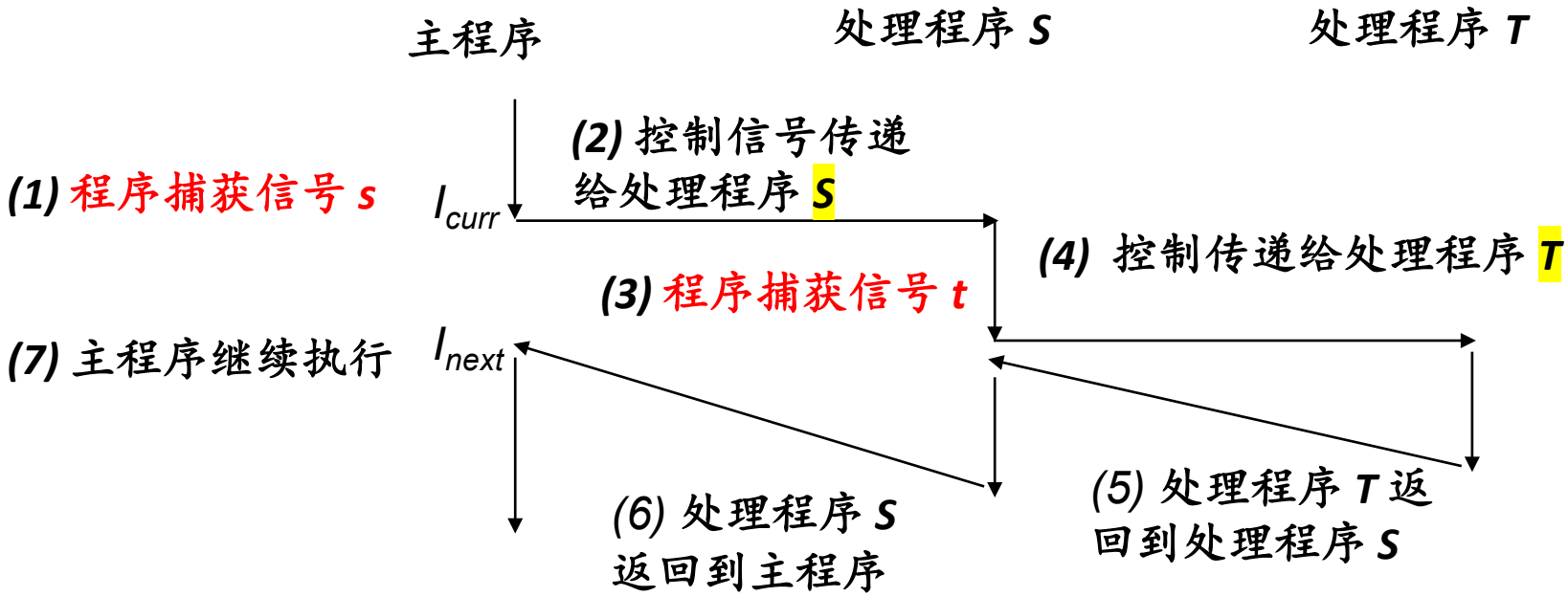
```
void sigint_handler(int sig)                /* SIGINT 信号处理程序 */
{
    printf("So you think you can stop the bomb with ctrl-c, "
           "do you?\n");
    sleep(2);                               // sleep 为 glibc 库函数
    printf("Well...");
    fflush(stdout);                         // 清空输出缓冲区
    sleep(1);
    printf("OK. :-)\n");
    exit(0);
}

int main()
{
    /* 设置 SIGINT 处理程序 */
    signal(SIGINT, sigint_handler);         // signal 为系统调用
    /* 等待接收一个信号 */
    Pause();                                // pause 为系统调用
    return 0;
}
```

换个视角看作为并发流的信号处理程序



信号处理程序的嵌套

- 信号处理程序可以被其它信号处理程序打断



信号的阻塞与解阻塞

- 隐式阻塞机制
  - 内核默认情况下会阻塞与当前正在处理信号类型相同的待处理信号
  - 例：一次对 SIGINT 信号的处理不能被另一个 SIGINT 信号打断
- 显式阻塞（解除阻塞）机制
  - sigprocmask 函数及其辅助函数可以显式地阻塞（解阻塞）选定的信号：SIG_BLOCK / SIG_UNBLOCK / SIG_SETMASK
- 辅助函数
  - sigemptyset 初始化 set 为空
  - sigfillset 把所有信号都添加到 set 中
  - sigaddset 把指定的信号 signum 添加到 set 中
  - sigdelset 从 set 中删除指定的信号 signum
  - sigismember 若 signum 在 set 中，则返回 1，否则返回 0