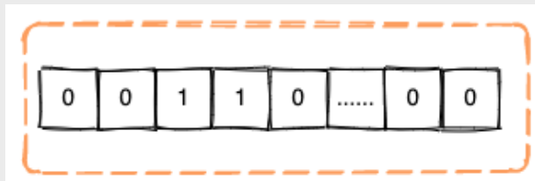
信号集与信号掩码

信号集

- 前面我们提到过，我们有 3 种常见处理信号的手段: 忽略，为其注册信号处理函数以及阻塞。阻塞某一个或者是某一类信号非常重要，比如说当我们使用信号处理函数处理某一个信号时，不希望其它的信号打断这个处理函数的执行
- 信号集用于表示某一个信号是否被阻塞，本质上就是一个数组所实现的 bitmap，使用结构体 sigset_t 表示



  - 既然是 bitmap 实现，那么每一种信号在 sigset_t 这一结构中只有两种状态: 被置位(1)，没有被置位(0)，来表示某一种信号是否被阻塞
  - sigemptyset —— 将 sigset_t 中全部的 bit 位置为 0，最为常用的初始化函数 —— sigset_t newset; sigemptyset(&newset);
  - sigfillset —— 将 sigset_t 中全部的 bit 位置为 1，也就是说阻塞所有信号的传递 —— sigset_t newset; sigfillset(&newset);
- 在初始化信号集之后，我们可以使用 sigaddset() 和 sigdelset() 这两个方法来对信号集进行添加和删除操作
  - int sigaddset(sigset_t *set, int sig);
  - int sigdelset(sigset_t *set, int sig);
- 除此之外，我们可以通过 sigismember() 来查询某个信号是否在信号集内 —— int sigismember(const sigset_t *set, int sig);

信号掩码

- **信号掩码的作用就是阻塞信号的传递，除了 SIGKILL 和 SIGSTOP 这两个必杀/必停的信号无法被阻塞以外，其余的信号都可以被阻塞**
- 内核会为每一个进程都维护一个信号掩码，也就是一组信号，阻塞其针对该进程的传递，知道进程从信号掩码中将该信号移除
- sigprocmask
  - 我们可以使用 sigprocmask() 系统调用，显式地向信号掩码中添加或移除信号，该函数主要接收参数为 sigset_t
  - int sigprocmask(int how, sigset_t *set, sigset_t *oddset); —— sigprocmask() 既可以设置新的掩码，也可以获取进程原有的掩码
  - **how**
    - SIG_BLOCK —— 将 set 指向信号集内的指定信号添加到信号掩码中。换言之，将信号掩码设置为其当前值 和 set 的并集
    - SIG_UNBLOCK —— 将 set 指向信号集中的信号从信号掩码中移除
    - SIG_SETMASK —— 将 set 指向的信号集赋给信号掩码
  - 当 oddset 不为空时，原有信号掩码将写入至该参数中
- 系统默认行为
  - 前面我们提到过，假设当前进程正在调用 SIGX 的信号处理函数，那么紧接着而来的 SIGX 信号将会被阻塞，直到上一个信号处理函数结束，这一现象我们可以用一个简单的例子证明

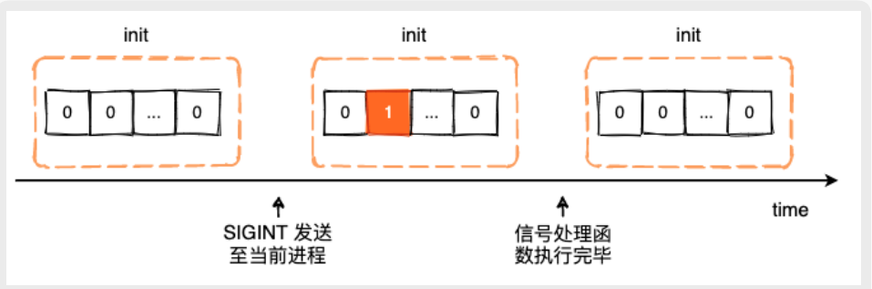
```
void handler(int signum) {
    printf("Got a SIGINT\n");

    sigset_t currentset;
    sigprocmask(SIG_BLOCK, NULL, &currentset);

    int res = sigismember(&currentset, SIGINT);

    printf("SIGINT is blocked ? : %d \n", res);
}
```

  - 我们用 SIGINT 作为捕获信号，当我们键入 Ctrl-C 时，将会发现 SIGINT 信号的确是在当前进程的信号掩码中



- **引发对处理器程序调用的信号将自动添加到进程信号掩码中。这意味着，当正在执行处理器程序时，如果同一个信号实例第二次抵达，信号处理器程序将不会递归中断自己**
- 处于等待状态的信号
  - 有时候我们想要知道当前进程阻塞了哪些信号，此时可以使用 sigpending() 来获得处于等待状态的信号集
  - int sigpending(sigset_t *set); —— 处于等待的信号集将会被置于形参 set 中
  - **实际上，当一个进程接收了被阻塞的信号时，那么会将该信号添加至等待信号集中，而等待信号集其实也是一个 sigset_t 结构**
- **等待信号集只是一个掩码，仅表明一个信号是否发生，而未表明其发生的次数。换言之， 如果同一信号在阻塞状态下产生多次，那么会将该信号记录在等待信号集中，并在稍后仅传递一次。也就是说，假设我们向进程发送了 1000 次 SIGINT 信号，进程可能仅接收 200 次**

sigaction

- signal() 函数由于本身有移植性的问题，所以该函数多用于 demo 演示中，不会在生产代码中出现，生产则使用 sigaction() 来改变信号的处置
- 基本原型
  - int sigaction(int sig, const struct sigaction *act, struct sigaction *oddact);

```
struct sigaction {
    void    (*sa_handler)(int);    // 信号处理函数指针
    sigset_t sa_mask;              // 信号掩码
    int      sa_flags;             // 信号处理函数调用时的控制
}
```

- sa_flags
  - sa_flags 字段是一个位掩码，指定用于控制信号处理过程的各种选项。该字段包含的位如下(可以相或(|))
  - SA_RESTART —— 自动重启由信号处理器程序中断的系统调用
- Example

```
int main() {
    sigset_t procmask;
    sigemptyset(&procmask);
    struct sigaction sa = {handler, procmask, SA_RESTART};
    sigaction(SIGINT, &sa, NULL);    // 忽略错误处理，并且并不关心原有 SIGINT 的处理情况
}
```