Lab4 挑战性任务

任务背景

信号（英语：Signal）是 Unix、类 Unix 以及其他 POSIX 兼容的操作系统中进程间通讯的一种有限制的方式。它是一种异步的通知机制，用来提醒进程一个事件已经发生。当一个信号发送给一个进程，操作系统会打断进程正常的控制流程，此时，任何非原子操作都将被打断。如果进程注册了信号的处理函数，那么它将被执行，否则就执行默认的处理函数。信号的机制类似于硬件中断（异常），不同之处在于中断由处理器发出并由内核处理，而信号由内核发出并由用户程序处理。除了进程通过系统调用向另一进程（或它自身）发出的信号，内核还可以将发生的中断通过信号通知给引发中断的进程。如果说系统调用是一种用户程序通知内核的机制，那么信号就是内核通知用户程序的机制。

你的任务是在我们的 MOS 操作系统中实现这样一套机制。

任务描述

本挑战性任务总共包含三个部分的内容，需要支持信号的**注册**、**发送**和**处理**。

信号的注册

信号的注册函数采用下面的函数：

```
int sigaction(int signum, const struct sigaction *act, struct sigaction
*oldact);
```

其中 `signum` 表示需要处理信号的编号，`act` 表示新的信号处理结构体，旧的信号处理结构体则需要在 `oldact != NULL` 时保存该指针在对应的地址空间中。

为了简化信号机制，你只需要考虑 `signum` 小于或等于 `64` 的情况，当收到编号大于64的信号时直接返回异常码 `-1`。

`sigaction` 结构体包含下面两个部分：

```
struct sigaction{
    void (*sa_handler)(int);
    sigset_t sa_mask;
};
```

其中 `sa_handler` 表示信号处理函数，`sa_mask` 表示在**运行信号处理函数过程中**的信号掩码，其结构体如下所示：

```
struct sigset_t{
    int sig[2]; //最多 32*2=64 种信号
};
```

信号的屏蔽是指当进程在收到信号时，不触发该信号的处理函数，而是等到信号屏蔽结束后才处理该信号。`sigset_t` 的每一位(bit)表示一种信号的屏蔽掩码，当其为 `0` 时不屏蔽该信号，当其为 `1` 时则屏蔽该信号直到其位置被修改为 `0`。

你需要实现下面的函数来对**进程的信号掩码**进行修改：

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

其中 `how` 表明对于信号掩码的修改类型方式，包括下面三种方式：

- `SIG_BLOCK`（`how` 为 `0`）：将 `set` 参数中指定的信号添加到当前进程的信号掩码中
- `SIG_UNBLOCK`（`how` 为 `1`）：将 `set` 参数中指定的信号从当前进程的信号掩码中删除
- `SIG_SETMASK`（`how` 为 `2`）：将当前进程的信号掩码设置为 `set` 参数中指定的信号集

当 `oldset` 不为 `NULL` 时，还需将原有的信号掩码放在 `oldset` 指定的地址空间中。

正常执行则返回 `0`，否则返回异常码 `-1`.

为了方便对于信号集操作，你需要实现下面五个函数：

```
void sigemptyset(sigset_t *set); // 清空信号集，将所有位都设置为 0
void sigfillset(sigset_t *set); // 设置信号集，即将所有位都设置为 1
void sigaddset(sigset_t *set, int signum); // 向信号集中添加一个信号，即将指定信号的位设置为 1
void sigdelset(sigset_t *set, int signum); // 从信号集中删除一个信号，即将指定信号的位设置为 0
int sigismember(const sigset_t *set, int signum); // 检查一个信号是否在信号集中，如果在则返回 1，否则返回 0
```

信号的发送

信号的发送采用下面的函数：

```
int kill(u_int envid, int sig);
```

向进程控制号编号为 `envid` 的进程发送 `sig` 信号，注意当 `envid = 0` 时代表向自身发送信号。该函数成为完成返回 `0`，如果信号编号超过限制或者进程编号不存在则返回 `-1`。

信号的处理

当进程被调度在之后继续运行时，首先需要处理自身收到的所有信号，对于后面来的信号，认为其优先级更高而先进行处理。对于被阻塞信号，**保留**其信号信息。对于未阻塞信号，则需要跳转到信号处理函数，如果未注册处理函数，则按照默认的处理动作进行。

本挑战性任务只考虑 `64` 种信号，其编号为从 `1` 到 `64`，每一个信号都对应信号掩码的一位。

信号	编号	来源	描述	默认处理动作
`SIGKILL`	9	操作系统	强制结束程序的运行（该信号不能被阻塞）	结束程序的运行
`SIGSEGV`	11	操作系统	用户程序访问了页表中未映射且地址严格小于 `0x3FE000` 的虚拟页	结束程序的运行
`SIGTERM`	15	用户程序	用于终止进程，但允许目标进程通信号处理函数拦截	结束程序的运行

其余类型信号默认处理动作为**忽略**。

注意事项

- 上述所说的函数声明均为用户态函数，部分函数需要通过系统调用陷入内核态完成修改
- 注意 `SIGSEGV` 信号的处理，不应该在内核态 panic
- 多个信号的处理需要实现信号重入机制，注意用户上下文的保存
- fork 后子进程应继承父进程的信号处理函数
- 注意将内核态的内容拷贝到用户态时，需要考虑该地址空间存在写时复制的问题
- 注意信号的编号为从1开始的下标编码

简单的测试程序

通过以下测试程序不能确保你的实现是正确的，请构造更多的测试程序。

基本信号测试

测试代码

```
#include <lib.h>

int global = 0;
void handler(int num) {
    debugf("Reach handler, now the signum is %d!\n", num);
    global = 1;
}

#define TEST_NUM 2
int main(int argc, char **argv) {
    sigset_t set;
    sigemptyset(&set);
    struct sigaction sig;
    sig.sa_handler = handler;
    sig.sa_mask = set;
    panic_on(sigaction(TEST_NUM, &sig, NULL));
    sigaddset(&set, TEST_NUM);
    panic_on(sigprocmask(0, &set, NULL));
    kill(0, TEST_NUM);
    int ans = 0;
    for (int i = 0; i < 10000000; i++) {
        ans += i;
    }
    panic_on(sigprocmask(1, &set, NULL));
    debugf("global = %d.\n", global);
    return 0;
}
```

正确输出

```
Reach handler, now the signum is 2!
global = 1.
```

空指针测试

测试代码

```
#include <lib.h>

int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *test = NULL;
void sgv_handler(int num) {
    debugf("Segment fault appear!\n");
    test = &a[0];
    debugf("test = %d.\n", *test);
    exit();
}

int main(int argc, char **argv) {
    sigset_t set;
    sigemptyset(&set);
    struct sigaction sig;
    sig.sa_handler = sgv_handler;
    sig.sa_mask = set;
    panic_on(sigaction(11, &sig, NULL));
    *test = 10;
    debugf("test = %d.\n", *test);
    return 0;
}
```

正确输出

```
Segment fault appear!
test = 1.
```

写时复制测试

测试代码

```
#include <lib.h>

sigset_t set2;

int main(int argc, char **argv) {
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set, 1);
    sigaddset(&set, 2);
    panic_on(sigprocmask(0, &set, NULL));
    sigdelset(&set, 2);
    int ret = fork();
    if (ret != 0) {
        panic_on(sigprocmask(0, &set2, &set));
        debugf("Father: %d.\n", sigismember(&set, 2));
    } else {
        debugf("Child: %d.\n", sigismember(&set, 2));
```

```
    }
    return 0;
}
```

正确输出

```
Father: 1.
Child: 0.
```

任务要求

- 你需要给出你的信号系统设计，报告各个信号结构体所在位置
- 对于各种类型的信号和使用场景，你应该给予充分的测试
- 请在lab6完成的基础上，基于lab6分支，自行建立 `lab4-challenge` 分支，在该分支完成代码后，push 到个人的远程仓库。代码内需要包含对于功能的详细测试程序，测试程序本身及运行测试程序得到的运行结果应具有足够的可读性。

  ```
  git checkout lab6
  git add .
  git commit -m "xxxxx"
  git checkout -b lab4-challenge
  # 完成代码
  git push origin lab4-challenge:lab4-challenge
  ```

- 实验报告请提交至spoc系统，在书写实验报告和准备申优答辩时，请加入以下内容：
- 对于任务的实现思路，并配合关键代码进行说明。
- 对于功能的详细测试程序，以及运行测试程序得到的运行结果。
- 完成挑战性任务过程中遇到的问题及解决方案。