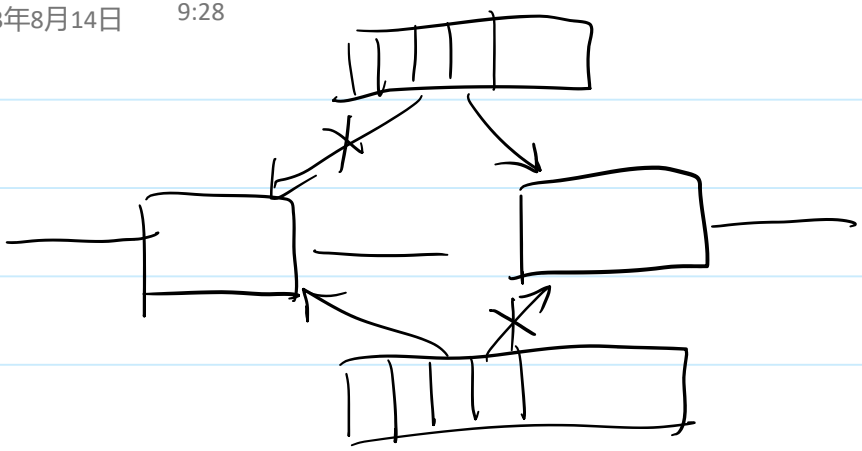
2023年8月14日 9:28



删除共享内存

2023年8月14日 10:25

```
$ ipcrm -m 15
$ ipcrm -a
```

延迟删除. 删除时若有进程连接到共享内存.

① key → 0

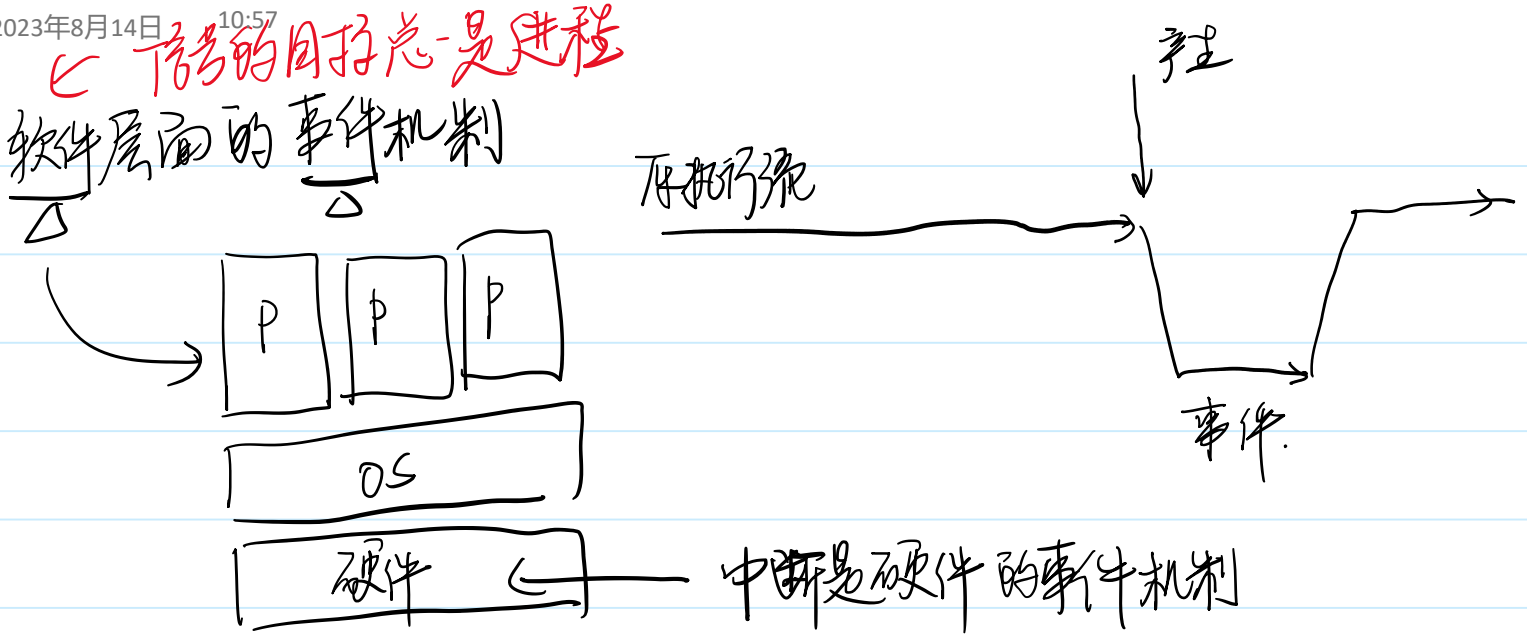
② status dest

当 nattach → 0, 真正删除

信号

2023年8月14日 10:57

信号的目标总是进程

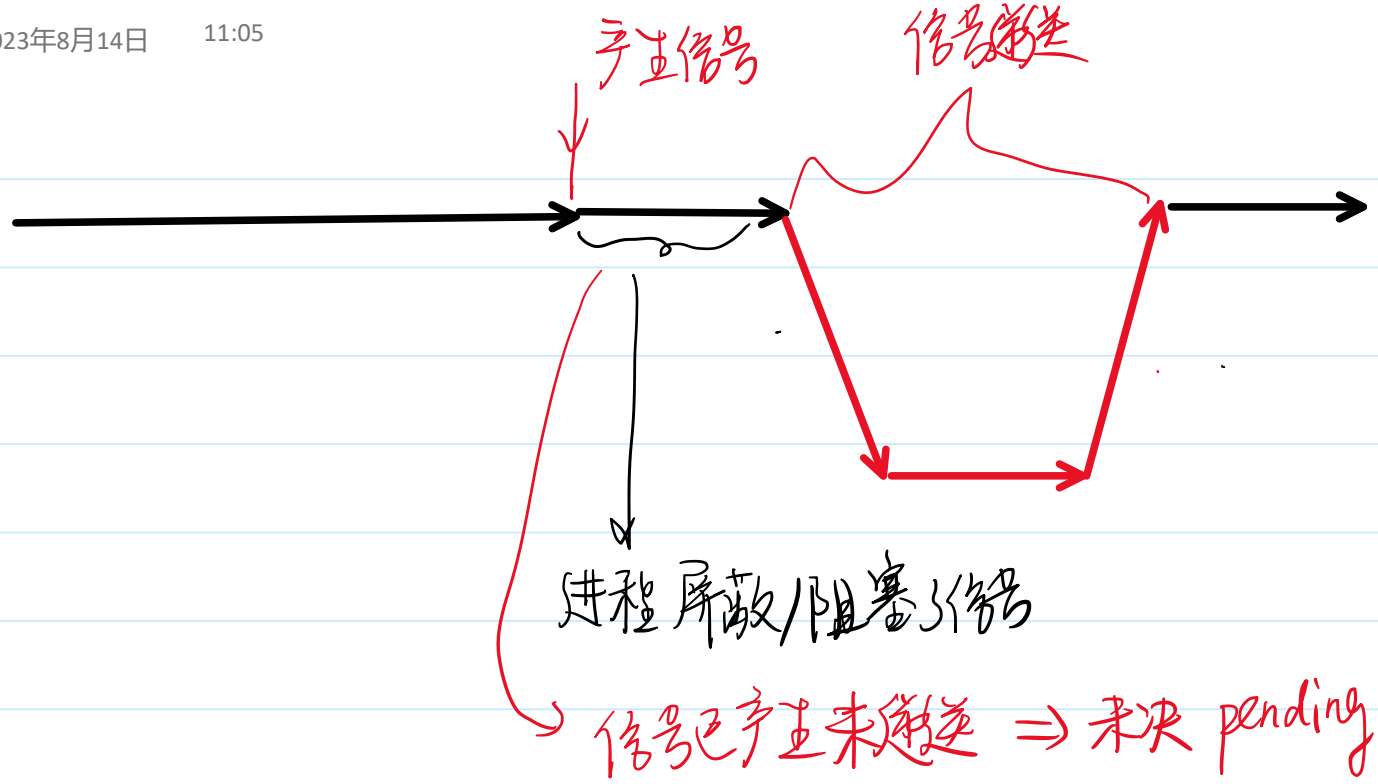


异步：事件的顺序是随机的. ← 信号是支持异步的.

同步：　　固定的.

信号处理阶段

2023年8月14日 11:05



产生一个信号

2023年8月14日 11:10

① 用户按下 ctrl+c ctrl+\ 硬件产生. 异步

② 代码执行出现异常 硬件产生 同步

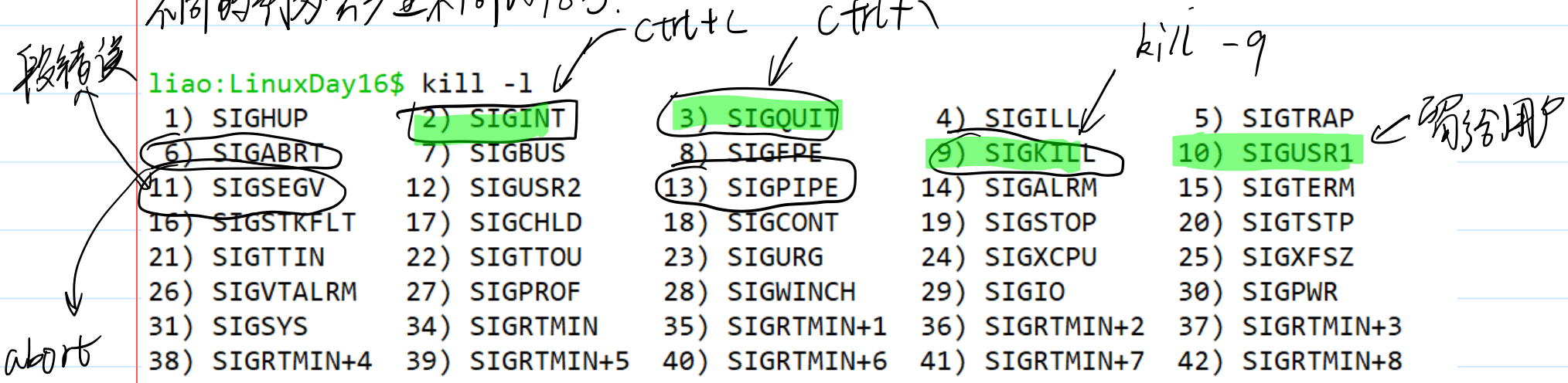
③ kill 软件产生 异步

④ abort 软件产生 同步

信号的分类

2023年8月14日 11:15

不同的行为会产生不同的信号.




```
liao:LinuxDay16$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

man 7 signal

2023年8月14日 11:16

Signal	Standard	Action	Comment
SIGABRT	P1990	Core	Abort signal from **abort**(3)
SIGALRM	P1990	Term	Timer signal from **alarm**(2)
SIGBUS	P2001	Core	Bus error (bad memory access)
SIGCHLD	P1990	Ign	Child stopped or terminated
SIGCLD	-	Ign	A synonym for **SIGCHLD**
SIGCONT	P1990	Cont	Continue if stopped
SIGEMT	-	Term	Emulator trap
SIGFPE	P1990	Core	Floating-point exception
SIGHUP	P1990	Term	Hangup detected on controlling terminal or death of controlling process
SIGILL	P1990	Core	Illegal Instruction
SIGINFO	-		A synonym for **SIGPWR**
SIGINT	P1990	Term	Interrupt from keyboard
SIGIO	-	Term	I/O now possible (4.2BSD)
SIGIOT	-	Core	IOT trap. A synonym for **SIGABRT**
SIGKILL	P1990	Term	Kill signal
SIGLOST	-	Term	File lock lost (unused)
SIGPIPE	P1990	Term	Broken pipe: write to pipe with no readers; see **pipe**(7)
SIGPOLL	P2001	Term	Pollable event (Sys V). Synonym for **SIGIO**
SIGPROF	P2001	Term	Profiling timer expired

递送

2023年8月14日 11:21

5种默认行为.

Term Default action is to terminate the process. 中止. ← SIGINT.

Ign Default action is to ignore the signal. 忽略 ← SIGCHLD

Core Default action is to terminate the process and dump core (see **core**(5)). 中止并生成core文件.

Stop Default action is to stop the process. 暂停

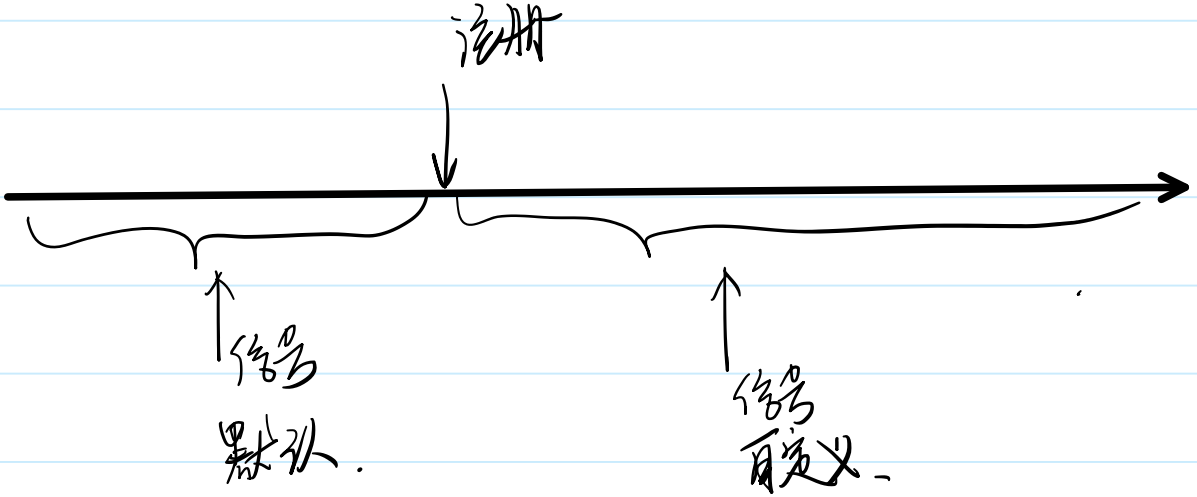
Cont Default action is to continue the process if it is currently stopped. 恢复.

```
liao:LinuxDay16$ ./3_signal
^C
liao:LinuxDay16$ ./3_signal
^\Quit (core dumped)
```

注册信号

2023年8月14日 11:29

修改信号的处理行为，不会执行。后续产生信号，才会执行。




```
void (*signal(int sig, void (*func)(int)))(int);
```

函数指针的类型别名

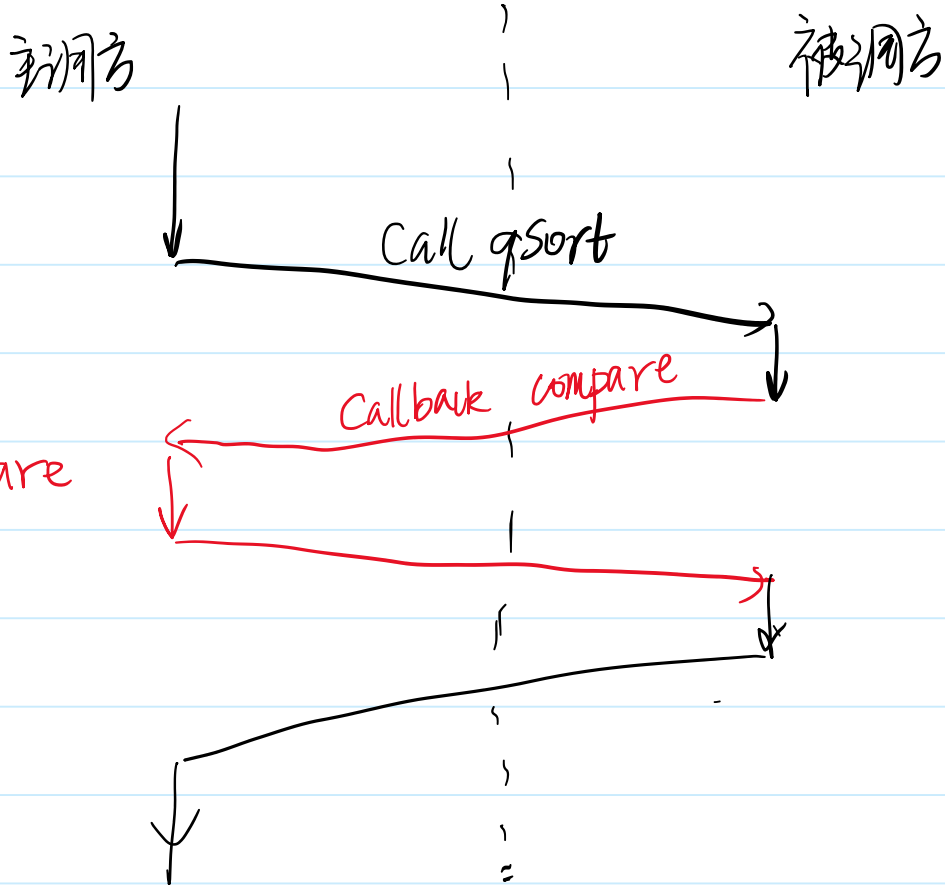
```
typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

signal 返回一个函数指针

signal 有两个参数 → 第1个参数 int
→ 第2个参数 函数指针

回调函数

2023年8月14日 11:31



回调函数：
主调方设计、被调方调用。

C语言实现 Callback，用到函数指针。

主调方设计、被调方调用：

```
int compare(:    ){}
```

qsort(---, compare)

函数指针

2023年8月14日 11:39

```c
void func(int arg){
    printf("arg = %d\n", arg);
}
int main(int argc, char *argv[])
{
    func(1);
    //void *pf(int arg); //这是函数声明 ()优先级高于*
    void (*pf)(int); // 离pf最低的是*，pf是一个指针
    pf = func;
    pf(1);
    return 0;
}
```

指针的练习题

2023年8月14日 11:53

```
int *p[3]; // [] 优先级高于 * p是一个数组
int (*p)[3]; // p是一个指针，指向一个长度为3的数组

void *p(int); // 函数声明
void (*p)(int); // p是一个函数指针变量
```

signal

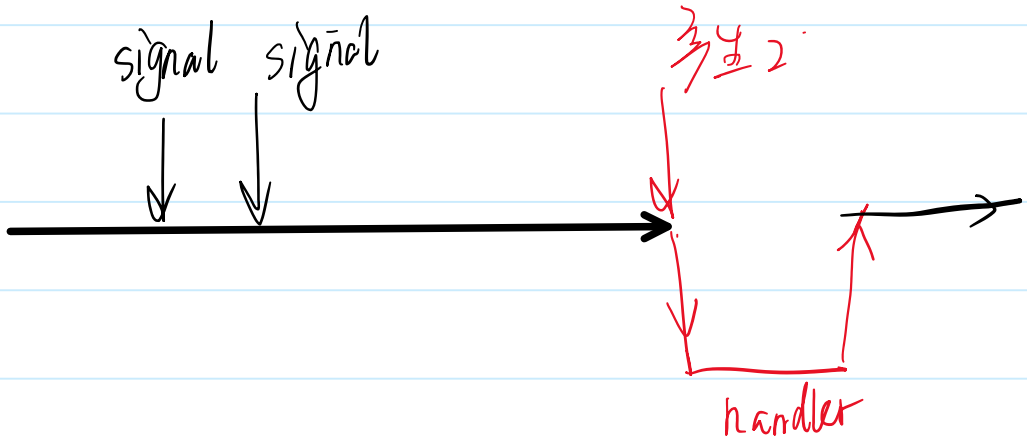
2023年8月14日 11:57

注册信号：修改处理行为

回调函数.

```
sighandler_t signal(int signum, sighandler_t handler);
```

```c
#include <52func.h>
void handler(int signum){
    printf("signum = %d\n", signum);
}
int main(int argc, char *argv[])
{
    signal(SIGINT,handler);//不会阻塞
    signal(SIGQUIT,handler);//不会阻塞
    while(1){
        sleep(1);
    }
    return 0;
}
```

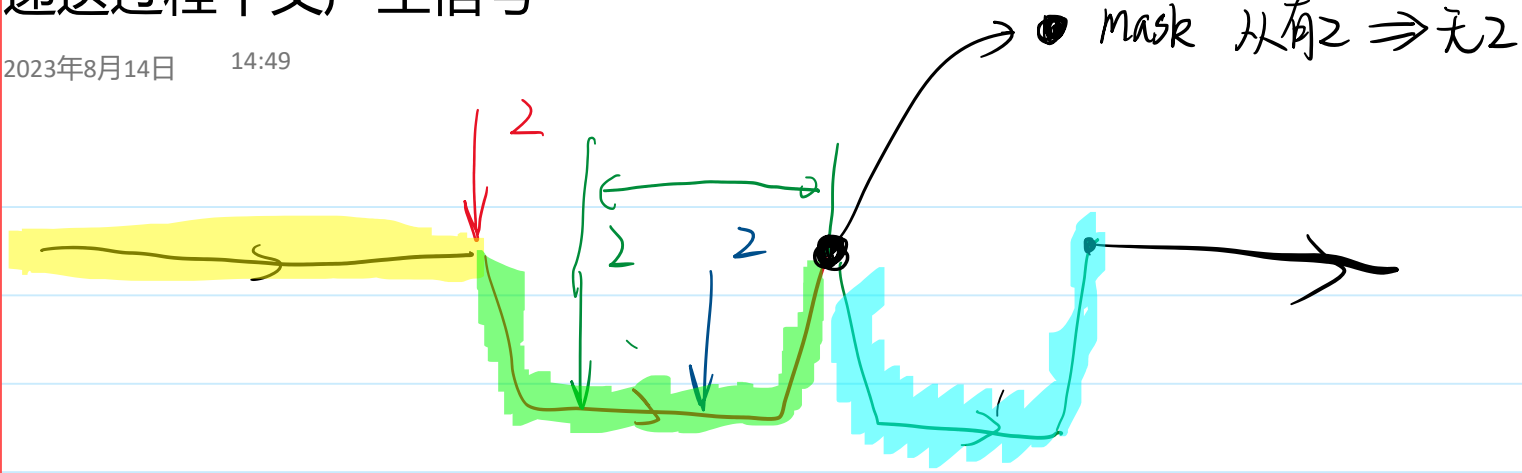


2023年8月14日 14:48

```c
void handler(int signum){
    printf("signum = %d\n", signum);
}
int main(int argc, char *argv[])
{
    signal(SIGINT,handler);
    sleep(10);
    printf("sleep over!\n");
    signal(SIGINT,SIG_DFL);//将信号的递送行为改成默认
    while(1){
        sleep(1);
    }
    return 0;
}
```

递送过程中又产生信号

2023年8月14日 14:49



mask为空. 产生任何信号都会立刻递达

mask中 2号信号存在

mask中 2号信号存在

在信号递达中, 使用signal注册的信号会临时屏蔽

不会 只有递达中.

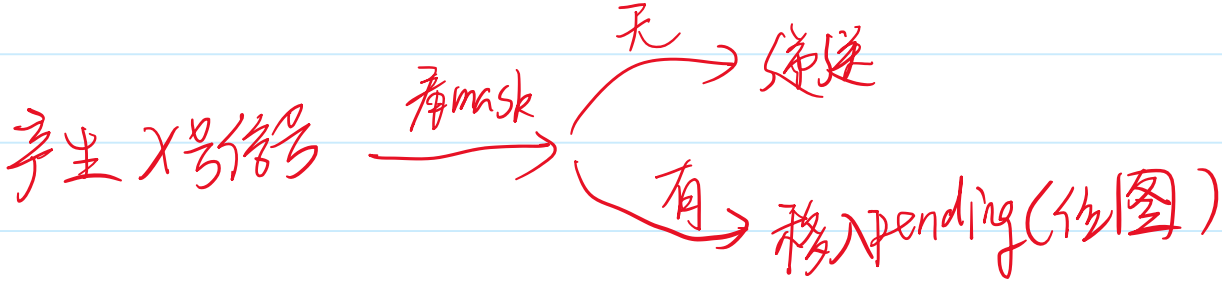
信号递送的核心数据结构

2023年8月14日 14:55

位图 ←— 每个信号1个bit.

mask. 掩码. 某个进程是否屏蔽了某个信号.

pending 存在未决信号. 每个信号只能保存"有"或"没有"



当 mask变化时, 从有X信号到无X信号, 看pending, 如果pending有X信号就取出来递送掉.

递送过程中产生一个别的信号

2023年8月14日 15:11

```
liao:LinuxDay16$ ./7_signal_different
^Cbefore signum = 2
^\before signum = 3
after signum = 3
after signum = 2
```

在递送x信号，只会临时屏蔽x，非XX屏蔽.

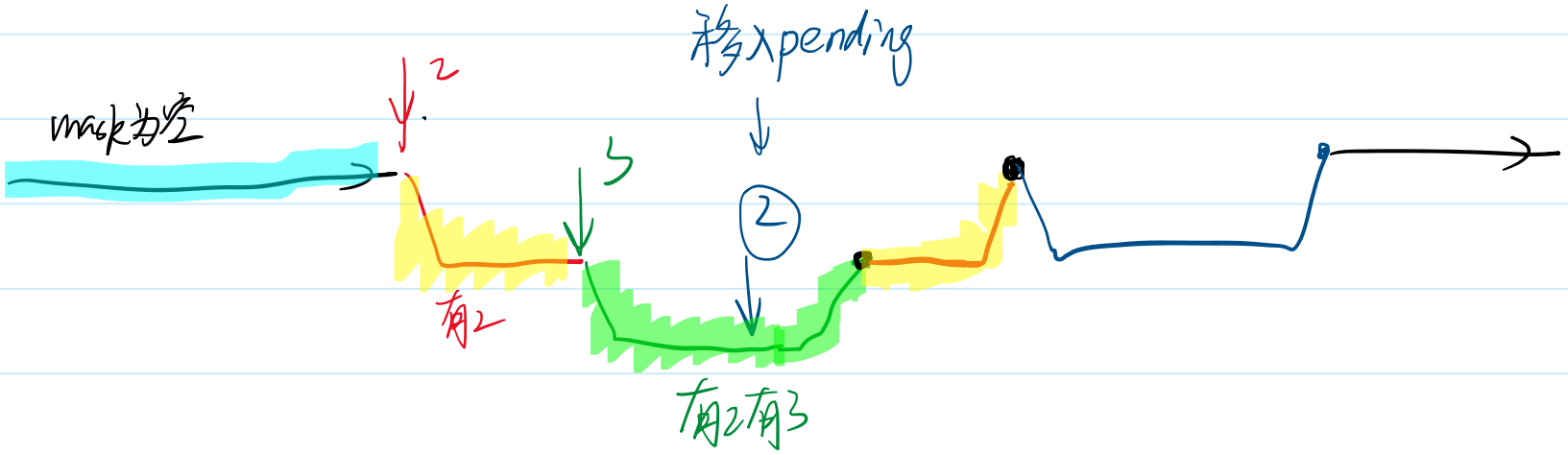
mask为空.

mask有2

mask有2也有3

2 3 2

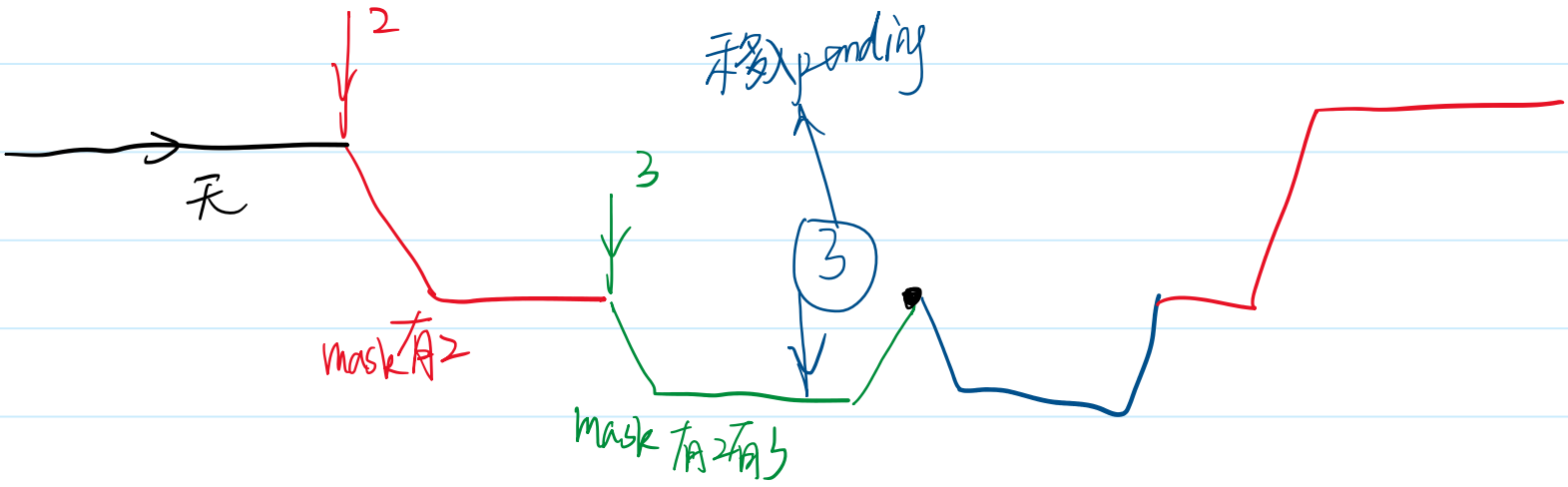
2023年8月14日 15:15




```
^Cbefore signum = 2
^\before signum = 3
^Cafter signum = 3
after signum = 2
before signum = 2
after signum = 2
```

2 3 3

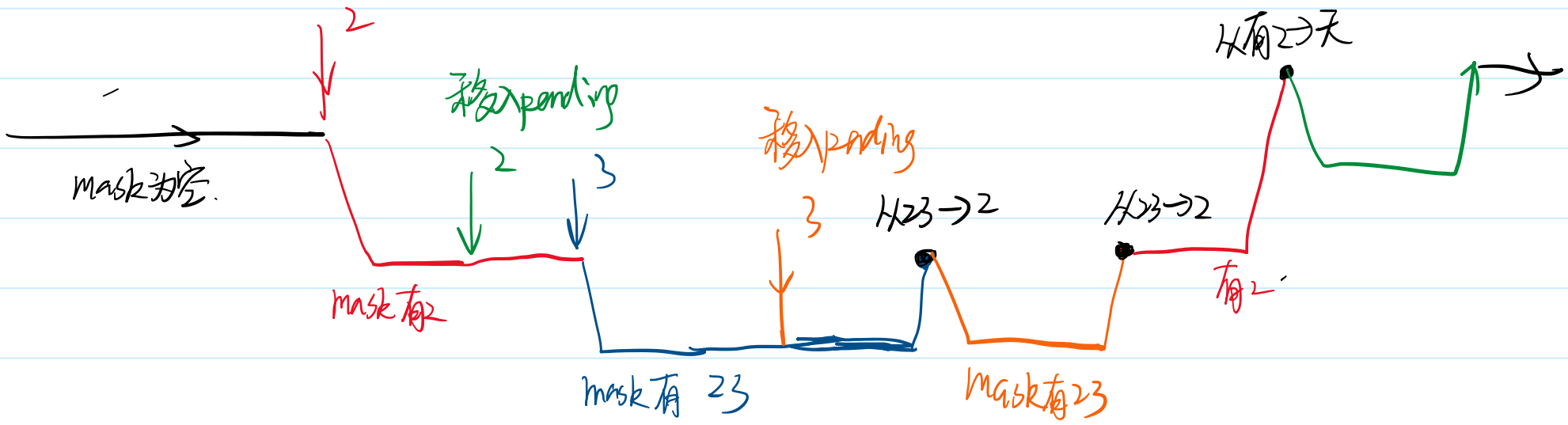
2023年8月14日 15:20




```
^Cbefore signum = 2
^\before signum = 3
^\after signum = 3
before signum = 3
after signum = 3
after signum = 2
```

2 2 3 3

2023年8月14日 15:25



整理一下signal的性质

2023年8月14日 15:51

1. 一次注册，永久生效。

2. 在递送信号时，会临时屏蔽本信号，不会屏蔽其他信号。

3. 递送完成以后，会自动重启低速系统调用。

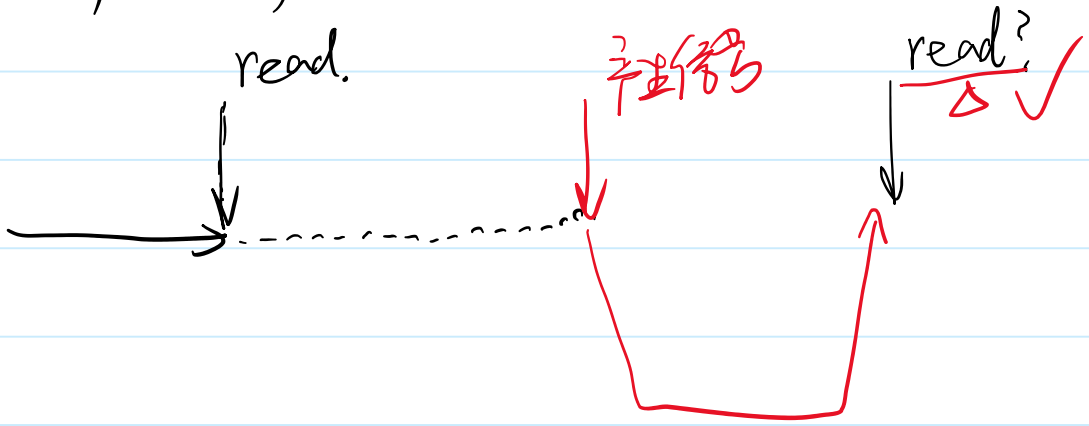
低速系统调用

2023年8月14日 15:54

有可能会陷入永久阻塞的系统调用.

read(STDIN_FILENO, ...)

read(fdr, ...)



进程因为低速系统调用而阻塞，信号被处理完成后，会自动重启.

2023年8月14日 16:04

```c
void handler(int signum){
    printf("signum = %d\n", signum);
}
int main(int argc, char *argv[])
{
    signal(SIGINT,handler);
    char buf[4096] = {0};
    ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
    printf("sret = %ld, buf = %s\n", sret,buf);
    //sleep(30);
    //printf("sleep over!\n");
    return 0;
}
```

sigaction

2023年8月14日 16:04

注册信号 比signal控制的属性更多.

```
#include <signal.h>

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

act 传入 要设置的新属性

oldact 传入传出. 保存旧的属性

NULL 不保存.

```
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    int        sa_flags;
    void     (*sa_restorer)(void);
};
```

两个回调函数只能存在一个

sa_mask → 额外临时屏蔽

sa_flags → 属性

sa_restorer(void); X

默认情况 flag 为0. 选 sa_handler 而不是 sa_sigaction

sigaction

2023年8月14日 16:11

```c
void handler(int signum){
    printf("before signum = %d\n", signum);
    sleep(5);
    printf("after signum = %d\n", signum);
}
int main(int argc, char *argv[])
{
    struct sigaction act;
    memset(&act,0,sizeof(act));
    act.sa_handler = handler;
    sigaction(SIGINT,&act,NULL);
    sigaction(SIGQUIT,&act,NULL);

    //while(1){
    //    sleep(1);
    //}
    char buf[4096] = {0};
    ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
    printf("sret = %ld, buf = %s\n", sret, buf);
    return 0;
}
```

① 一次注册永久生效

② 处理过程中，临时屏蔽本信号
不会屏蔽其他信号

③ 不会自动重启低速系统调用.

sigaction的其他属性

2023年8月14日 16:21

SA_RESTART　　sa_flags 存在，自动重启低速系统调用.

SA_RESETHAND　　一次注册，一次生效

SA_NODEFER　　不再临时屏蔽本信号.

```
//act.sa_flags = SA_RESTART|SA_RESETHAND;
act.sa_flags = SA_RESTART|SA_NODEFER;
```

位图的操作

2023年8月14日 16:28

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);
```

清空.

每个bit置1.

增加一个

移除一个

查找信号是否在集合中.

额外临时屏蔽 sa_mask

2023年8月14日 16:39

```c
void handler(int signum){
    printf("before signum = %d\n", signum);
    sleep(5);
    printf("after signum = %d\n", signum);
}
int main(int argc, char *argv[])
{
    struct sigaction act;
    memset(&act,0,sizeof(act));
    act.sa_handler = handler;
    //act.sa_flags = SA_RESTART|SA_RESETHAND;
    act.sa_flags = SA_RESTART;
    sigemptyset(&act.sa_mask); //清空集合
    sigaddset(&act.sa_mask,SIGQUIT);
    sigaction(SIGINT,&act,NULL);
    sigaction(SIGQUIT,&act,NULL);
```

原来：

递送X时，屏蔽X，不屏蔽Y.

现在 sa_mask有Y.

递送X时，屏蔽Y.

递送Y时，屏蔽又Y，不屏蔽X.

sa_mask 间接影响mask

sigaction 3参数的回调函数

2023年8月14日 17:06

sa_flags 一般包含

```
SA_SIGINFO (since Linux 2.2)
       The  signal  handler takes three arguments, not one.
siginfo_t {
    int      si_signo;     /* Signal number */
    int      si_errno;     /* An errno value */
    int      si_code;      /* Signal code */
    int      si_trapno;    /* Trap number that caused
                              hardware-generated signal
                              (unused on most architectures) */
    pid_t    si_pid;       /* Sending process ID */
    uid_t    si_uid;       /* Real user ID of sending process */
```

```c
void new_handler(int signum, siginfo_t *info, void *ucontext){
    printf("signum = %d\n", signum);
    printf("pid = %d, uid = %d\n", info->si_pid, info->si_uid);
}
int main(int argc, char *argv[])
{
    struct sigaction act;
    memset(&act,0,sizeof(act));
    act.sa_sigaction= new_handler;
    act.sa_flags = SA_RESTART|SA_SIGINFO;
    sigemptyset(&act.sa_mask); //清空集合
    sigaddset(&act.sa_mask,SIGQUIT);
    sigaction(SIGINT,&act,NULL);
```

sigpending

2023年8月14日 17:17

获取pending集合. 已产生未递送.

功能: 递送2号. 额外屏蔽3号.

2号递送完成之时, 查询pending中有没有3号

```
void handler(int signum){
    printf("before, signum = %d\n", signum);
    sleep(10);
    printf("after, signum = %d\n", signum);

    sigset_t pending;
    sigpending(&pending);
    if(sigismember(&pending,SIGQUIT)){
        printf("SIGQUIT is pending!\n");
    }
    else{
        printf("SIGQUIT is not pending!\n");
    }
}
int main(int argc, char *argv[])
{
    struct sigaction act;
    memset(&act,0,sizeof(act));
    act.sa_handler=handler;
    act.sa_flags = SA_RESTART;
    sigemptyset(&act.sa_mask);
    sigaddset(&act.sa_mask,SIGQUIT);

    sigaction(SIGINT,&act,NULL);
```

```
int sigpending(sigset_t *set);
```

mask的变化 sigprocmask

2023年8月14日 17:28

信号开始 mask 从无X，→有X
信号结束 mask 从有X→无X
} 信号 临时屏蔽.

永久屏蔽

新的set 原来的set

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

+ SIG_BLOCK oldset 并上 set
  The s
- SIG_UNBLOCK oldset 移除 set
  The s
  is nc
= SIG_SETMASK 用set赋值

在递送过程之外增加或者解除屏蔽

2023年8月14日 17:35

```c
int main(int argc, char *argv[])
{
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set,SIGQUIT);

    sigprocmask(SIG_BLOCK,&set,NULL);

    sleep(10);
    printf("sleep over!\n");

    sigprocmask(SIG_UNBLOCK,&set,NULL);
    return 0;
}
```

kill raise

2023年8月14日 17:40

```c
int main(int argc, char *argv[])
{
    // ./13_kill pid
    ARGS_CHECK(argc,2);
    pid_t pid = atoi(argv[1]);
    kill(pid,SIGKILL);
    return 0;
}
```

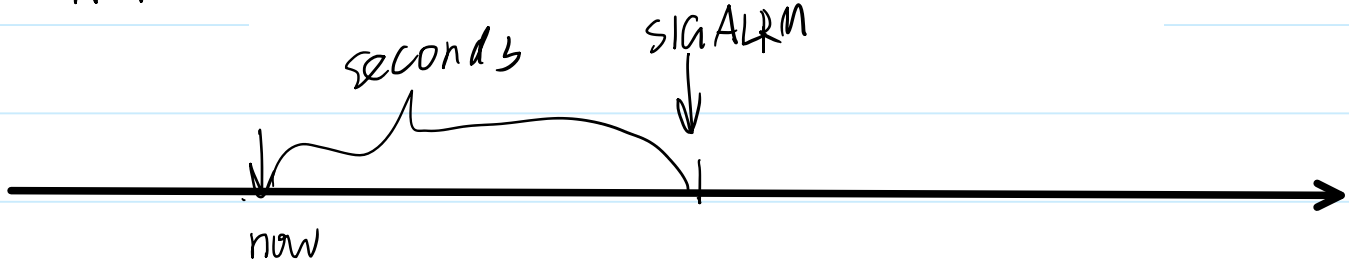
```c
int raise(int sig);
```

alarm

2023年8月14日 17:46

定闹钟.

```
unsigned int alarm(unsigned int seconds);
```




```c
void handler(int signum){
    printf("signum = %d\n", signum);
    time_t now = time(NULL);
    printf("curtime = %s\n", ctime(&now));
}
int main(int argc, char *argv[])
{
    handler(0);
    signal(SIGALRM,handler);
    alarm(10);
    while(1){
        sleep(1);
    }
    return 0;
}
```

pause

2023年8月14日 17:51

```
NAME
       pause - wait for signal

SYNOPSIS
       #include <unistd.h>

       int pause(void);
```

pause 导致进程阻塞，直到 任一信号 被处理完成

```
alarm(10);
pause();
```
⟹ sleep.