# Sigaction 挑战性任务报告


> 姓名：文柳懿
> 学号：21351002


## 结构体定义

- 信号是一种进程间通讯ipc，因此信号的信息要放在进程控制块内。
- Env结构体

```
struct Env {
    /*...*/
    //lab4-challenge
    //携带的信息
    struct sigaction env_sigaction[33]; //信号集
    struct sigset_t env_sa_mask; //处理掩码
    struct Sig_wait_list sig_wait_list;//等待列表
    //函数栈
    int running_sig[33];//正在执行的信号处理 函数栈
    int env_sig_top;//正在处理的信号栈顶，如果栈为空为-1

    u_int env_user_signal_func; //用户处理函数
}
```

### sigset_t,sigaction结构体

```
//32位表示MOS所需要处理的[1,32]信号掩码：对应位为1表示阻塞，为0表示未被阻塞
typedef struct sigset_t {
    uint32_t sig;
} sigset_t;

struct sigaction {
    void     (*sa_handler)(int);    //处理函数
    sigset_t   sa_mask;     //对应信号被处理时，需要被屏蔽的信号集
};
```

### 信号处理结构体

```
struct Env_signal{
    int signum;
    TAILQ_ENTRY(Env_signal) sig_wait_link;
    LIST_ENTRY(Env_signal) sig_free_link;
};
TAILQ_HEAD(Sig_wait_list,Env_signal);   //存放当前进程：等待接收的信号结构体
LIST_HEAD(Sig_free_list,Env_signal); //存放空闲的信号处理结构体
```

# 函数定义

```
int sigemptyset(sigset_t *__set);
int sigfillset(sigset_t *__set);
int sigaddset(sigset_t *__set, int __signo);
int sigdelset(sigset_t *__set, int __signo);
int sigismember(const sigset_t *__set, int __signo);
int sigisemptyset(const sigset_t *__set);
int sigandset(sigset_t *__set, const sigset_t *__left, const sigset_t *__right);
int sigorset(sigset_t *__set, const sigset_t *__left, const sigset_t *__right);
int sigprocmask(int __how, const sigset_t * __set, sigset_t * __oset);
int sigpending(sigset_t *__set);

//系统调用函数：进入内核态
int syscall_sigaction(int signum, const struct sigaction *act, struct sigaction
*oldact);
int syscall_sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
int syscall_kill(u_int envid, int sig);
void syscall_set_env_user_signal_func(u_int envid,u_int func);
void syscall_pop_running_sig();
```

# 函数实现

## 1. 信号注册

- `sigaction` -> `syscall_sigaction`

> `sigaction`函数

```
/*
    signum:需要设置的信号编号；
    newact:为signum设置的sigaction结构体,如果newact不为空；
```

```
    oldact: 将该信号之前的sigaction结构体其内容，填充到oldact中(如果oldact不为空)
*/
int sigaction(int signum, const struct sigaction *newact, struct sigaction
*oldact){
    if(signum<1 || signum>32) return -1;
    return syscall_sigaction(signum,act,oldact);
}
```

`syscall_sigaction`函数

```
int syscall_sigaction(int signum, const struct sigaction *newact, struct sigaction
*oldact){
    return msyscall(SYS_sigaction, signum, newact, oldact);
}
```

`sys_sigaction`函数

```
int sys_sigaction(int signum, const struct sigaction *act, struct sigaction
*oldact){
    //如果oldact不为空:将该信号之前的sigaction结构体其内容填充到oldact中
    if(oldact!=NULL) *oldact=curenv->env_sigaction[signum];
    curenv->env_sigaction[signum].sa_handler=act->sa_handler;
    curenv->env_sigaction[signum].sa_mask.sig=act->sa_mask.sig;
}
```

## 2. 信号掩码

- `SIG_BLOCK`：将 `__set` 中指定的信号，添加到当前进程的信号掩码中
- `SIG_UNBLOCK`：从当前进程的信号掩码中，移除`__set`中指定的信号
- `SIG_SETMASK`：将当前进程的信号掩码，设置为 `__set`中指定的信号集

`sigprocmask`函数,`sys_sigprocmask`函数

```
int sigprocmask(int how, const sigset_t *__set, sigset_t *__oset){
    return syscall_sigprocmask(how,s__set,__oset);
}
int sys_sigprocmask(int how, const sigset_t *set, sigset_t *oldset){
    if(oldset!=NULL){
        *oldset=curenv->env_sa_mask; //注意是要赋值而不是给赋上地址
    }
    if(how==SIG_BLOCK){
        curenv->env_sa_mask.sig[0]|=set->sig[0];
        curenv->env_sa_mask.sig[1]|=set->sig[1];
```

```
    }
    else if(how==SIG_UNBLOCK){
        curenv->env_sa_mask.sig[0]&=~set->sig[0];
        curenv->env_sa_mask.sig[1]&=~set->sig[1];
    }
    else if(how==SIG_SETMASK){
        curenv->env_sa_mask.sig[0]=set->sig[0];
        curenv->env_sa_mask.sig[1]=set->sig[1];
    }
    return 0;
}
```

## 3.信号集操作

```
//清0:清空参数的__set掩码
int sigemptyset(sigset_t *__set){
    __set->sig=0;
}

//全为1:将参数中的__set掩码填满
int sigfillset(sigset_t *__set){
    __set->sig=0xffffffff;  //32位
}

//置位为1:向__set信号集中添加一个信号__signo。如果操作成功，__set将包含该信号。
int sigaddset(sigset_t *__set, int __signo){
    __set->sig |= 1<<__signo;
}

//置位为0:从__set信号集中删除一个信号__signo。如果操作成功，__set将不再包含该信号。
int sigdelset(sigset_t *__set, int __signo){
    __set->sig &= ~(1<<__signo);
}

//检查信号__signo是否是__set信号集的成员:如果是，返回1；如果不是，返回0
int sigismember(const sigset_t *__set, int __signo){
    return __set->sig & (1<<__signo);
}

// 检查信号集__set是否为空。如果为空，返回1；如果不为空，返回0。
int sigisemptyset(const sigset_t *__set){
    return (__set->sig==0);
}

// 计算两个信号集__left和__right的交集，并将结果存储在__set中。
int sigandset(sigset_t *__set, const sigset_t *__left, const sigset_t *__right){
```

```
    uint32_t left=(uint32_t)__left;
    uint32_t right=(uint32_t)__right;
    __set->sig = left & right;
}

// 计算两个信号集__left和__right的并集，并将结果存储在__set中。
int sigorset(sigset_t *__set, const sigset_t *__left, const sigset_t *__right){
    uint32_t left=(uint32_t)__left;
    uint32_t right=(uint32_t)__right;
    __set->sig = left | right;
}
```

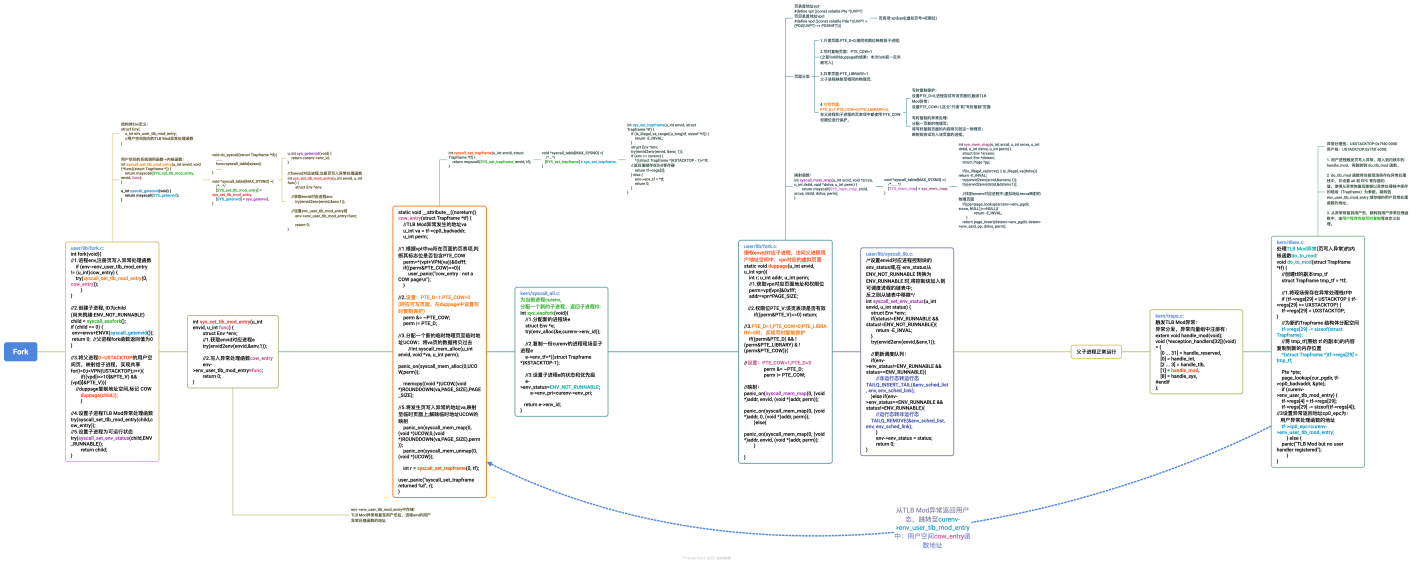
## 4. 信号发送

```
int kill(u_int envid, int sig){
    if(!(sig>=1&&sig<=32)) return -1;
    return syscall_kill(envid,sig);
}

int sys_kill(u_int envid, int sig){
    printk("%s\n","sys_kill");
    struct Env *e;
    if(envid2env(envid,&e,0)<0) return -1;  //对应进程不存在

    if(LIST_EMPTY(&sig_free_list)) return -1;
    //从空闲列表中获取一个结构体，然后将其清空并设置信号值为 sig
    struct Env_signal *signal=LIST_FIRST(&sig_free_list);
    memset(signal,0,sizeof(struct Env_signal));
    signal->signum=sig;

    //if(e->env_user_signal_func==0) sys_set_env_user_signal_func(e->env_id,
sighandler);
    /*
        #define TAILQ_FOREACH(var, head, field)
\
            for ((var) = ((head)->tqh_first); (var); (var) = ((var)-
>field.tqe_next))
    */
    struct Env_signal *signal_existed;

    TAILQ_FOREACH(signal_existed,&e->sig_wait_list,sig_wait_link){
        if(sig<signal_existed) {    //编号小的信号，优先级高
            TAILQ_INSERT_BEFORE(signal_existed, signal, sig_wait_link);
            return 0;
        }else if(sig==signal_existed){  //同一普通信号在进程中最多只存在一个
            return 0;
```

```
        }
    }
    TAILQ_INSERT_TAIL(&e->sig_wait_list,signal,sig_wait_link);//插在队列末尾
    return 0;
}
```

# 5.信号处理

- 思想：模仿写时复制机制



### 注册信号处理函数

```
static void __attribute__((noreturn)) sighandler(struct Trapframe *tf,int num,void
(*sa_handler)(int)) {
    if(sa_handler){
        void (*func)(int);
        func=sa_handler;
        func(num);
        syscall_pop_running_sig();
        syscall_set_trapframe(0,tf);
    }
    else if(num==SIGSEGV||num==SIGKILL||num==SIGTERM){
        syscall_pop_running_sig();
        exit(); //结束进程
    }
}
```

### 异常分发

- `kern/genex.S` 增加：
  `BUILD_HANDLER signal do_signal`

- `kern/traps.c`：

```
extern void handle_signal(void);
//异常向量组中注册handle_signal
void (*exception_handlers[32])(void) = {
#if !defined(LAB) || LAB >= 4
    [1] = handle_mod,
    [8] = handle_sys,
    [10] = handle_signal,
#endif
};
```

**异常处理函数**

```
void do_signal(struct Trapframe *tf){
    struct Env_signal *signal;
    TAILQ_FOREACH(signal,&curenv->sig_wait_list,sig_wait_link){
        if(signal!=NULL&&signal->signum>=1&&signal->signum<=32){
            if(signal->signum==SIGSEGV||signal->signum==SIGKILL){
                //直接处理
            }
            else if(curenv->env_sig_top>-1){ //表明在一个信号处理函数中
                u_int signum=curenv->running_sig[curenv->env_sig_top];
                int mask=curenv->env_sigaction[signum].sa_mask.sig;

                if(((mask>>((signal->signum-1)%32))&0x1)){
                    continue;
                }
            }
            else{
                int mask=curenv->env_sa_mask.sig;
                if(((mask>>((signal->signum-1)%32))&0x1)){
                    continue;
                }
            }
            //处理该信号
            TAILQ_REMOVE(&curenv->sig_wait_list,signal,sig_wait_link);
            if (curenv->env_sigaction[signal->signum].sa_handler||
                signal->signum==SIGSEGV||signal->signum==SIGKILL||
                signal->signum==SIGINT||signal->signum==SIGILL) {
                struct Trapframe tmp_tf = *tf;
                if (tf->regs[29] < USTACKTOP || tf->regs[29] >= UXSTACKTOP) {
```

```
                    tf->regs[29] = UXSTACKTOP;
                }
                tf->regs[29] -= sizeof(struct Trapframe);
                *(struct Trapframe *)tf->regs[29] = tmp_tf;
                //if(!curenv->env_user_signal_func)
syscall_set_env_user_signal_func(curenv->env_id, sighandler);
                if(curenv->env_user_signal_func){
                    tf->regs[4] = tf->regs[29];
                    tf->regs[5] = signal->signum;
                    tf->regs[29] -= sizeof(tf->regs[4]);
                    tf->regs[29] -= sizeof(tf->regs[5]);
                    tf->regs[29] -= sizeof(tf->regs[6]);
                    sys_push_running_sig(signal->signum);
                    tf->cp0_epc = curenv->env_user_signal_func;
                }
            }
            break;
        }
    }
}
```