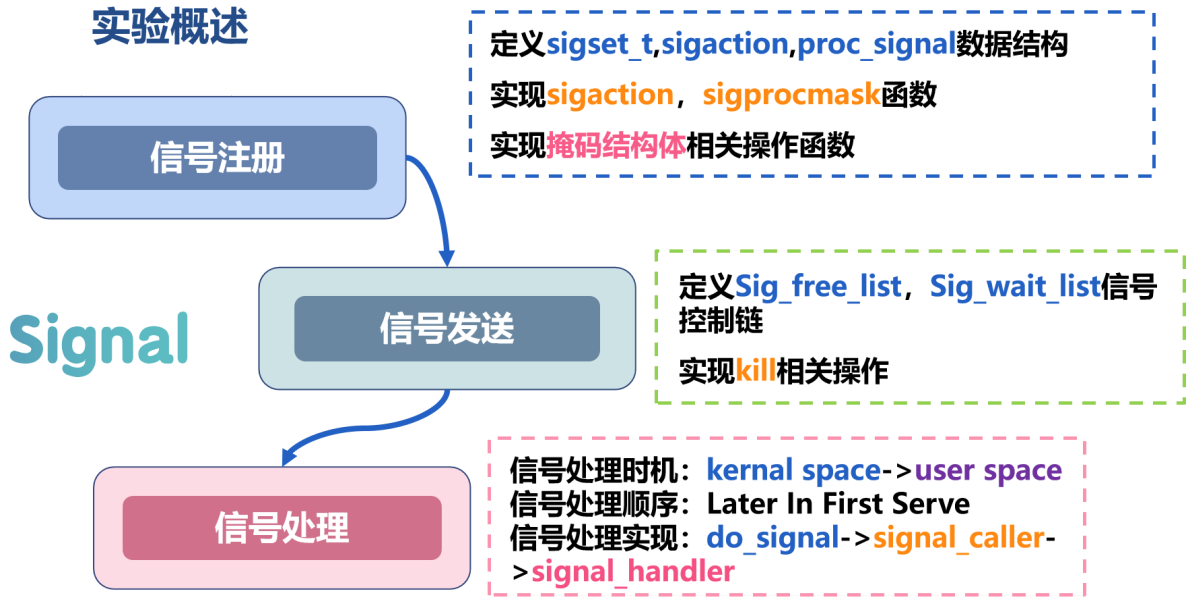
Lab4-challenge实验报告

> Lab4-challenge旨在要求我们实现一个信号机制，支持信号的**注册**、**发送**和**处理**。



信号注册

相关数据结构准备

增加signal.h

在本实验中，要求定义了两个结构体，`sigaction`和`sigset`，分别作为信号的处理函数以及掩码，我在内核中`user/include`添加了`signal.h`文件，将结构体定义在其中，实现如下：

```
//signal.h
#ifndef __SIGNAL_H__
#define __SIGNAL_H__

#define SIG_BLOCK 0
#define SIG_UNBLOCK 1
#define SIG_SETMASK 2
#define SIGKILL 9
#define SIGSEGV 11
#define SIGTERM 15

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

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

TAILQ_HEAD(Sig_wait_list,proc_signal);
LIST_HEAD(Sig_free_list,proc_signal);
```

```
struct proc_signal{
    int signum;
    TAILQ_ENTRY(proc_signal) sig_wait_link;
    LIST_ENTRY(proc_signal) sig_free_link;
};

#endif
```

除此以外，我定义了Sig_wait_list，Sig_free_list以及proc_signal结构体，具体作用会在之后细讲。

在这里需要注意的是，**定义内核中的.h文件，需要在文件的前后加上以下字样，才能保证该文件被引用时不会因为多次引用而报重复定义的错误。**

```
#ifndef __XXXXX_H__//文件名
#define __XXXXX_H__

#endif
```

修改env.h

信号机制可以通过`sigaction`方法为当前进程注册信号处理函数，因此，我们需要有相应的数据结构去存储当前进程所所注册的信号。同时，信号为我们提供了屏蔽信号的机制，因此我们需要为进程添加全局的信号处理掩码。于是，在`env.h`中，我添加了

```
struct sigaction env_sigaction[65]; //信号处理函数集,指针数组
struct sigset_t env_sigset_t;       //进程的信号屏蔽掩码
```

函数实现

准备好相应的数据结构，我们就可以实现相应的函数了。在`user/lib`下我填了`signal.c`文件，用于相关函数的添加，函数实现如下：

```
//signal.c
#include<lib.h>
/*
清空信号集，将所有位都设置为 0
*/
void sigemptyset(sigset_t *set){
    set->sig[0]=0;
    set->sig[1]=0;
}
/*
清空信号集，将所有位都设置为 1
*/
void sigfillset(sigset_t *set){
    set->sig[0]=0xffffffff;
    set->sig[1]=0xffffffff;
}
/*
向信号集中添加一个信号，即将指定信号的位设置为 1
*/
void sigaddset(sigset_t *set, int signum){
    set->sig[(signum-1)/32]|=1<<((signum-1)%32);
```

```
}
/*
从信号集中删除一个信号，即将指定信号的位设置为 0
*/
void sigdelset(sigset_t *set, int signum){
    set->sig[(signum-1)/32]&=~(1<<((signum-1)%32));
}
/*
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
*/
int sigismember(const sigset_t *set, int signum){
    if((set->sig[(signum-1)/32]>>((signum-1)%32))&0x1){
        return 1;
    }
    else{
        return 0;
    }
}
/*
信号注册函数
*/
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)
{
    /*
    signum的范围是1~64,不满足范围返回-1。
    */
    if(!(signum>=1&&signum<=64)){
        return -1;
    }
    return syscall_sigaction(signum,act,oldact);
}
/*
修改进程的信号掩码
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
*/
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset){
    return syscall_sigprocmask(how,set,oldset);
}
```

在用户库中添加新的.c文件时，需要在/user/include.mk中添加signal.o，以便于该文件的成功编译。

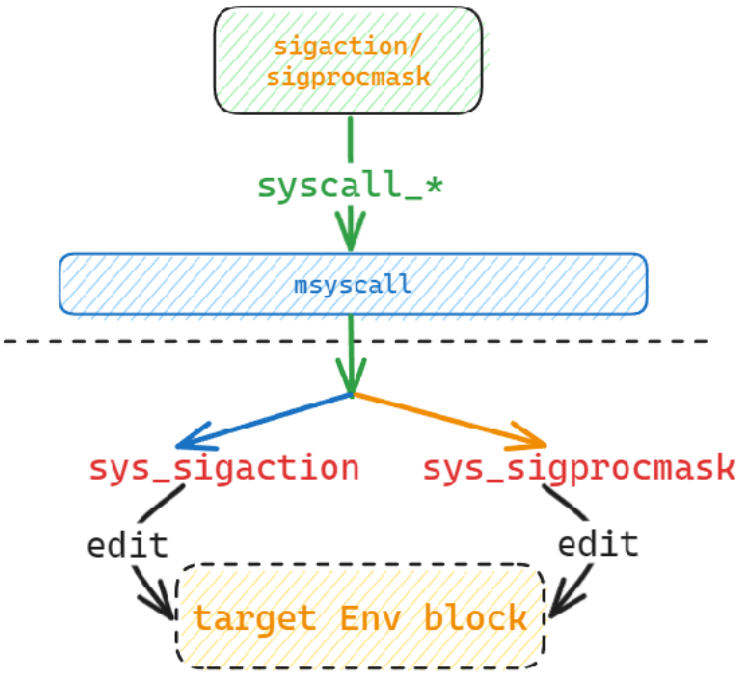
对于信号掩码的处理通过简单的**位运算处理**即可。由于`sigaction`和`sigprocmask`需要修改进程控制块中的内容，因此我选择使用**系统调用**进行处理，实现方式和lab4上机的方式相同，所以这里我只放出sys_*的具体实现：

```
int sys_sigaction(int signum, const struct sigaction *act, struct sigaction
*oldact){
    if(oldact!=NULL){
      *oldact=curenv->env_sigaction[signum]; //注意是要赋值而不是给赋上地址
    }
    curenv->env_sigaction[signum].sa_handler=act->sa_handler;
    curenv->env_sigaction[signum].sa_mask.sig[0]=act->sa_mask.sig[0];
    curenv->env_sigaction[signum].sa_mask.sig[1]=act->sa_mask.sig[1];
```

```
    return 0;
}
int sys_sigprocmask(int how, const sigset_t *set, sigset_t *oldset){
    if(oldset!=NULL){
        *oldset=curenv->env_sigset_t; //注意是要赋值而不是给赋上地址
    }
    if(how==SIG_BLOCK){//0
      curenv->env_sigset_t.sig[0]|=set->sig[0];
      curenv->env_sigset_t.sig[1]|=set->sig[1];
    }
    else if(how==SIG_UNBLOCK){//1
      curenv->env_sigset_t.sig[0]&=~set->sig[0];
      curenv->env_sigset_t.sig[1]&=~set->sig[1];
    }
    else if(how==SIG_SETMASK){//2
      curenv->env_sigset_t.sig[0]=set->sig[0];
      curenv->env_sigset_t.sig[1]=set->sig[1];
    }
    return 0;
}
```

此时，我们调用`sigaction`和`sigprocmask`，就可以进行信号的注册以及进程掩码的修改了。这个时候信号的注册就完成了。

以下是信号注册以及进程掩码注册的流程图：



信号发送

对于信号的发送，我们需要考虑两个问题，**第一个是怎么发，第二个是发了存在哪。**

数据结构准备

对于信号而言，并不是所有的信号刚收到就会立即处理，因此，对于信号所收到信号需要进行相应存储，这就设计到以什么数据结构表示信号，以及用什么方式存储信号的问题，在这里，我设计了`proc_signal`结构体用来表示信号，同时仿照`Env`结构体的存储和处理方式，定义了`Sig_wait_list`和`Sig_free_list`，分别表示进程的信号等待对列和全局的空闲信号链表。

增加proc_signal结构体

在`signal.h`中，我定义了`proc_signal`结构体，用作表示信号的数据结构，其中`signum`代表**当前结构体所表示的信号**，`TAILQ_ENTRY(proc_signal)`，`LIST_ENTRY(proc_signal)`是为了将该结构体作为`Sig_wait_list`和`Sig_free_list`，这里参照了`Env`结构体中的实现。

```
struct proc_signal{
    int signum;
    int sequence;
    TAILQ_ENTRY(proc_signal) sig_wait_link;
    LIST_ENTRY(proc_signal) sig_free_link;
};
```

设置Sig_wait_list

`Sig_wait_list`表示每个进程当前等待的信号链表，每个进程都有其相应的等待对列，因此我们需要把该链加入到Env结构体中。在这里我使用了**TAILQ**的结构，便于在链表的头部和尾部插入。在env_alloc申请得到进程控制块时，对其使用**TAILQ_INIT**初始化即可。

```
//env.h
struct ENV{
……
struct Sig_wait_list sig_wait_list;//等待信号列表
}
```

设置Sig_free_list

在为进程sig_wait_list中添加新的信号结构体时，需要考虑在**内核空间中申请固定的空间**来存储`proc_signal`结构体，因此我们就不能在信号发送时使用临时生成的局部变量。

对于这个问题曾经想过两个方法，第一个方法是使用`alloc`函数，MOS已经为我们提供了这个方法，虽然通过这个方法能够实现内存的分配，但是没有配备相应的free方法，因此通过这个方法在多等待信号被释放时，相应的空间得不到释放，导致内存泄漏，显然是不可取的。

那如何解决问题呢？Env的控制块的申请给了我启示，我可以在内核空间首先申请一定数量的`proc_signal`块，将其组织为空闲链表存在`sig_free_list`中，每次需要时从里面取出一块，回收时只需要插入原链表中，这样就实现了有限的`proc_signal`块复用，由于`sig_free_list`是**全局可见**的，我们只需要定义一次，因此我在env.c文件中，添加了以下结构：

```
//env.c
struct proc_signal signals[256] __attribute__((aligned(BY2PG)));
struct Sig_free_list sig_free_list;
```

在`env_init`函数中，也仿照对于进程控制块的处理，首先初始化`sig_free_list`，接下来将`proc_signal`数组中的每一块插入`sig_free_list`，之后就可以调用**LIST_FIRST**和**LIST_REMOVE**从中取出空闲信号块了。

```
//env.c
void env_init(){
    ……
    LIST_INIT(&sig_free_list);//初始化sig_free_list
    ……
    for(i=0;i<256;i++){//初始化signals，并插入链表中
        signals[i].signum=0;
        memset(&signals[i],0,sizeof(signals[i]));
        LIST_INSERT_HEAD((&sig_free_list),&signals[i],sig_free_link);
    }
}
```

函数实现

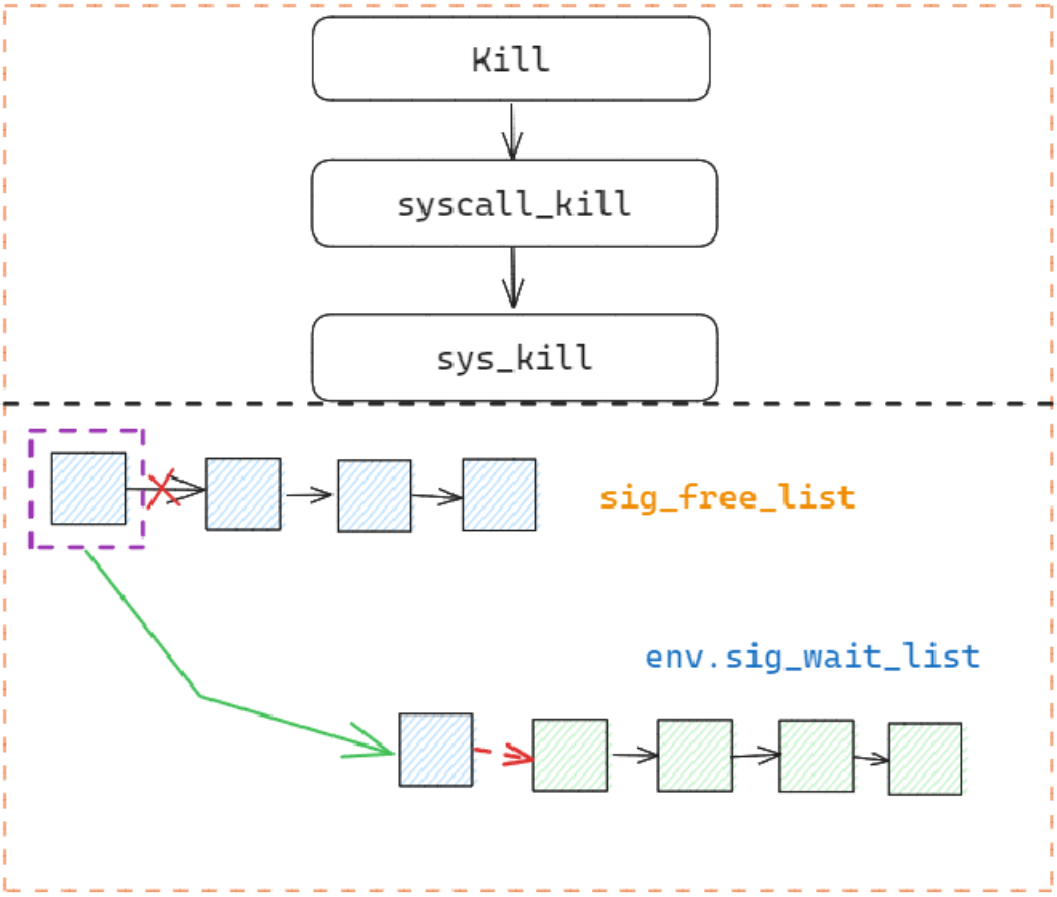
上面处理好了存在哪的问题，现在开始处理怎么发的问题。

我定义了**kill(u_int envid, int sig)**用来发送信号给目标进程，因为要涉及到修改目标进程的进程控制块，因此要使用系统调用。

```
//signal.c
int kill(u_int envid, int sig){
    if(!(sig>=1&&sig<=64)){
        return -1;
    }
    return syscall_kill(envid,sig);
}
```

```
//syscall_all.c
int sys_kill(u_int envid, int sig){
    struct Env *e;
    if(!(sig>=1&&sig<=64)){
      return -1;
    }
    if(envid2env(envid,&e,0)<0){ //关闭父子进程验证
        // printk("%d\n",envid);
        return -1;
    }
    e->is_running_add=1;
    if(LIST_EMPTY(&sig_free_list)){
      return -1;
    }
    struct proc_signal *signal=LIST_FIRST(&sig_free_list);
    LIST_REMOVE(signal,sig_free_link);
    memset(signal,0,sizeof(struct proc_signal));
    signal->signum=sig;
    TAILQ_INSERT_HEAD(&e->sig_wait_list,signal,sig_wait_link);//后来先处理
    return 0;
}
```

在**sys_kill**函数中，首先调用`envid2env`获取进程控制块，但是要关闭父子进程验证，因为可能是非父子进程间发送信息。接下来，从`sig_free_list`中取出一个`signal`块，设置好相关参数，由于是**后来先处理**的模式，将新来信号块**插入到TAILQ的头部**，取出时从头部取出，这样就完成了信号的发送。



信号处理

完成了完成了信号注册与信号发送，这时我们就要开始思考**何时处理信号**，**如何处理信号**的问题，接下来就以这两个方面入手。

信号处理时机

教程中要求我们修改**passive_alloc**函数中的第一个部分，要求注释掉内核中的panic，因此对于**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**的处理是操作系统向当前进程发送一个`SIGSEGV`的信号，由于是在内核，操作系统可以直接调用sys_kill(0,SIGSEGV)进行异常处理。

```
//tlbex.c
static void passive_alloc(u_int va, Pde *pgdir, u_int asid) {
  struct Page *p = NULL;
  if (va < UTEMP) {
    sys_kill(0,11);
    //panic("address too low");
  }
……
}
```

因此，如果仅在进程被调度时处理信号，可能会导致此类信号来不及立刻处理，所以**信号的处理应该被放在从内核返回用户态时**，对于相关函数修改而言，**进程切换，系统调用和异常处理在返回用户态时都会调用ret_from_exception**，因此就可以在这里作为信号处理的时机，仿照`do_tlb_mod`，我实现了一个`do_signal`，以此作为信号处理从内核到用户的跳板，我修改了`kern/genex.S`，让ret_from_exception时处理信号，

```
//genex.S
```

```
.text
FEXPORT(ret_from_exception)
    move    a0, sp # 给do_siganl传入参数
    addiu   sp, sp, -8
    nop
    jal     do_signal
    nop
    addiu   sp, sp, 8
    RESTORE_SOME
    lw      k0, TF_EPC(sp)
    lw      sp, TF_REG29(sp) /* Deallocate stack */
.set noreorder
    jr      k0
    rfe
.set reorder
```

主要增加了**RESTORE_SOME**前面这6行，作为调用进行处理，这样在每次从内核态返回用户态的时候就可以进行信号处理了。

signal_caller

signal_caller实现

处理好了信号处理时机，那现在应该想想，信号应该怎么去处理呢？在MOS中，**内核是不能直接调用用户的程序的**，因为这也是基于对于系统的保护，**因此我们需要回到用户态，在用户态中进行信号处理。**在用户态应该以什么样的形式去处理呢？

如果是直接回到当前信号相应的处理函数，虽然简单粗暴，但是**信号处理函数结束后不会再陷入内核恢复第一次进入内核时上下文的状态**，导致虽然处理了信号，进程的上下文却被破坏了，这种方法不可取。

但是对于tlb_mod的处理也是放在用户态处理的，那么就可以仿照tlb_mod的处理，仿照**cow_entry**的方法，在**用户态设置一个signal_caller**作为信号处理时返回的用户态的函数，在从内核态返回时，不论什么信号，都统一返回**signal_caller**，在其中再调用相应的信号处理函数进行处理，并且仿照**cow_entry**的实现，在**signal_caller**中**实现上下文恢复的功能**，因此在从内核返回用户时，我们需要为`signal_caller`传入需要恢复的**Trapframe**，**信号num**以及**相应的处理函数指针**，实现如下：

```
//libos.c
static void __attribute__((noreturn)) signal_caller(struct Trapframe *tf,int
num,void (*sa_handler)(int)) {
    if(sa_handler){
        void (*func)(int);
        func=sa_handler;
        func(num);
        syscall_set_trapframe(0,tf);//陷入内核态，恢复进程上下文。
    }
    else if(num==SIGSEGV||num==SIGKILL||num==SIGTERM){
        syscall_pop_running_sig();
        debugf("The process ended successfully.\n");
        exit(); //直接结束进程
    }
}
```

在调用完相应的信号处理函数之后，调用`syscall_set_trapframe(0,tf)`就可以**陷入内核态，恢复原进程的上下文，使进程正常运行。**

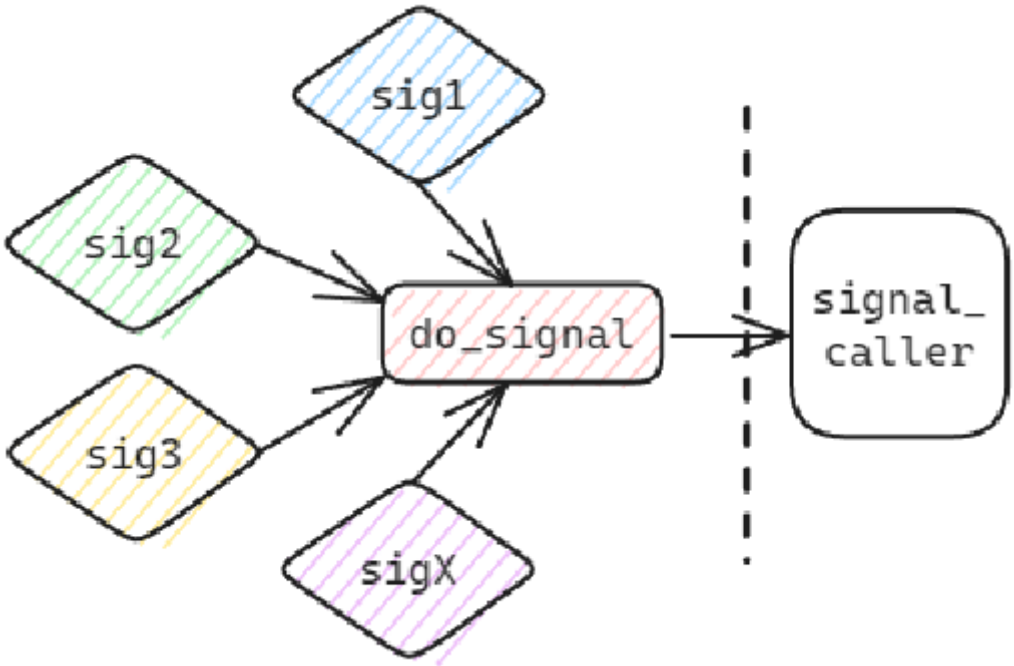
signal_caller注册

因为要在内核返回`signal_caller`函数，所以内核中需要有`signal_caller`的地址，就需要给进程注册`signal_caller`函数。在tlb_mod的**cow_entry**中，由于只会在`fork`之后产生写时复制，因此**只需要在fork的同时为进程注册cow_entry**，但是对于信号机制而言，我可能没有注册信号处理函数，但是操作系统会给进程发送SIGSEGV等信号，那么就**需要在进程启动伊始就注册**`signal_caller`,为进程准备好信号处理的机制，在指导书中得知，进程会从`libos.c`中启动`main`函数，因此，**只需要在启动**`main`**函数之前就为进程注册号**`signal_caller`**即可**，因此，我增加了`syscall_set_env_signal_caller`系统调用，在`libos.c`调用：

```
//libos.c
void libmain(int argc, char **argv) {
    syscall_set_env_signal_caller(0,signal_caller); //为进程注册signal_caller
    // set env to point at our env structure in envs[].
    env = &envs[ENVX(syscall_getenvid())];
    // call user main routine
    main(argc, argv);
    // exit gracefully
    exit();
}
```

```
//syscall_all.c
void sys_set_env_signal_caller(u_int envid,u_int func){
    struct Env *e;
    if(envid2env(envid,&e,0)<0){ //关闭父子进程验证
        panic("wrong wrong.\n");
    }
    e->env_signal_caller=func;
}
```

这样就可以完成`signal_caller`的注册。



实现do_signal

`do_signal`的实现如下：

```
//tlbex.c
void do_signal(struct Trapframe *tf){
  struct proc_signal *signal;
  TAILQ_FOREACH(signal,&curenv->sig_wait_list,sig_wait_link){
    if(signal!=NULL&&signal->signum>=1&&signal->signum<=64){
        if(signal->signum==SIGSEGV||signal->signum==SIGKILL){
            //直接处理
        }
        else if(curenv->env_sig_top>-1){
            u_int signum=curenv->running_sig[curenv->env_sig_top];
            int mask=curenv->env_sigaction[signum].sa_mask.sig[(signal->signum-
1)/32];
            if(((mask>>((signal->signum-1)%32))&0x1)){
                continue;
            }
        }
        else{
            int mask=curenv->env_sigset_t.sig[(signal->signum-1)/32];
            if(((mask>>((signal->signum-1)%32))&0x1)){
              continue;
            }
        }
        TAILQ_REMOVE(&curenv->sig_wait_list,signal,sig_wait_link);
        signal_handle(signal->signum,tf);
        signal->signum=0;
        LIST_INSERT_HEAD(&sig_free_list,signal,sig_free_link);
        break;
    }
  }
}
```

`do_signal`**函数的逻辑是选取第一个不被阻塞的信号进行处理。**因此我使用了**TAILQ_FOREACH的方法进行遍历。**接下来，首先要判断信号是不是特殊信号，比如**SIGSEGV**，**SIGKILL**这类不可以被屏蔽的信号，如果是就理解处理即可，接下来，判断此刻是否正在运行的信号处理函数中，我为每个进程维护了一个信号处理栈，如果即将进入信号处理函数，我就会将信号num加入到栈中，当信号处理函数结束并恢复现场时，我会将处理完的信号弹出栈中。为此，我实现了两个函数`sys_push_running_sig(num)`，`sys_pop_running_sig()`：

```
//syscall_all.c
void sys_push_running_sig(int signum){
    curenv->env_sig_top++;
    curenv->running_sig[curenv->env_sig_top]=signum;
    // printk("push %d\n",curenv->env_sig_top);
}

void sys_pop_running_sig(){
    if(curenv->env_sig_top>=0){
      curenv->running_sig[curenv->env_sig_top]=0;
      curenv->env_sig_top--;
      // printk("pop %d\n",curenv->env_sig_top);
```

```
    }
}
```

其中**压栈**，在`signal_handle`函数中，因为下一步就是跳转到`signal_caller`；**弹栈**方面，我为了减少从用户态陷入内核的次数，我在`sys_set_trapframe`中调用`sys_pop_running_sig()`，在恢复进程上下文的同时弹栈。

```
//syscall_all.c
int sys_set_trapframe(u_int envid, struct Trapframe *tf) {
    ……
  struct Env *env;
  try(envid2env(envid, &env, 1));
  if (env == curenv) {
    ……
    sys_pop_running_sig();//恢复进程上下文的同时弹栈
    ……
  } else {
    ……
  }
}
```

如果**curenv->env_sig_top**大于-1，说明当前正在信号处理函数中，对于新来的信号需要用**当前信号处理函数的掩码进行判断**，

如果**curenv->env_sig_top**为-1，说明当前无信号处理函数正在运行，根据进程的全局信号掩码进行阻塞判断即可。

如果成功取到信号块，需要将其从当前进程的sig_wait_list中取出，并将其放回sig_free_lits回收空间。

实现signal_handle

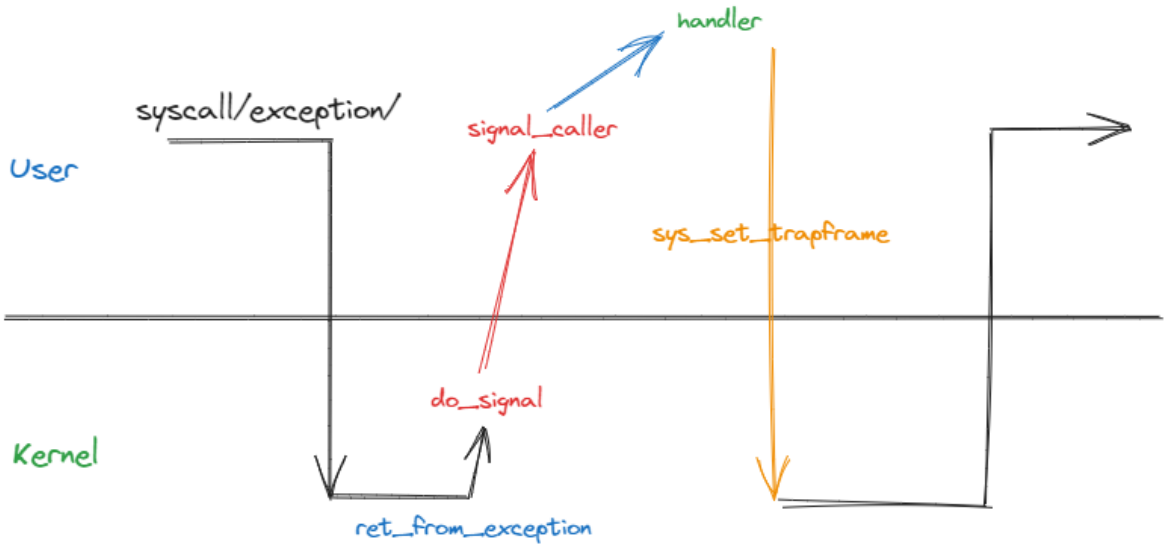
为了避免函数过于冗长，我将为处理函数开辟用户栈的过程封装成了`signal_handle`函数，其中仿照`do_tlb_mod`函数**为用户的信号处理函数在用户异常栈上开辟空间**，这里也实现了信号重入的机制，接下来，通过**设置当前栈帧的相关寄存器为signal_caller传递参数，并将cp0_epc设置为signal_caller的地址。**此时，跳转所需的条件都已经设置好了，别忘了，**do_signal是从ret_from_exception跳转过来的**，接下来就会**返回那里，跳转到用户态caller并执行信号处理函数，最后再返回内核设置原进程的上下文**，此时，信号的处理就已经完成了。

```
//tlbex.c
void signal_handle(int num,struct Trapframe *tf) {
  if (curenv->env_sigaction[num].sa_handler||
      num==SIGSEGV||num==SIGKILL||num==SIGTERM) {
    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_signal_caller){
      tf->regs[4] = tf->regs[29];
      tf->regs[5] = num;
      tf->regs[6] = curenv->env_sigaction[num].sa_handler;
      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(num);
        tf->cp0_epc = curenv->env_signal_caller;
        }
    } else {
      //默认不处理
    }
}
```

流程图

最终，整个信号的处理过程可以用以下的流程图表示：



遇到的问题

固定空间分配问题

一开始我是在sys_kill的函数中设置局部变量并插入到sig_wait_list中，但是由于内存在函数结束后就会释放，所以导致这样错误。之后我仿照env的操作为proc_signal设置了一个空闲链，实现固定内存的合理使用。

进程初始化问题

signal_caller未注册前是不能处理信号的，因此我在libos.c中首先注册了signal_caller，以保证进程开始就能实现信号处理机制。同时要注意的是，在env结构体中新加入的变量都要在env_alloc时进行初始化，对于全局的sig_free_list要在env_init中同env_sched_list和env_free_list一同初始化。

fork父子进程继承问题

在父子进程进行相关测试的时候，发现子进程不能处理信号，最后发现是因为子进程没有继承父进程的signal_caller导致出错，修改之后即可正常使用。

写时复制问题

在从内核态写入时，有可能会触发写时复制，但是在内核态是不能处理的，因此在向内核传入指针进行修改时，在用户态首先对指针进行修改，以便如果触发tlb_mod异常也会在用户态处理，这样就能规避了这个问题。

相关测试

Lab4_challenge_1

```
#include <lib.h>
#define TEST_NUM 10
/*
期望输出
sigaddset TEST_OK.
sigdelset TEST_OK.
You have changed register func for sig10.
register successfully!
You have passed register Test for sig10!
*/
void handler(int num){
    debugf("You have passed register Test for sig%d!\n",num);
}

void handler1(int num){
    debugf("You have changed register func for sig%d.\n",num);
}

int main(){
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set,3); //添加id为3的信号
    sigaddset(&set,5); //添加id为5的信号
    struct sigaction sig;
    struct sigaction swap_sig;
    if(sigismember(&set,3)==0){ //检查3是否在
        user_panic("wrong in sigaddset_1\n");
    }
    if(sigismember(&set,5)==0){ //检查5是否在
        user_panic("wrong in sigaddset_2\n");
    }
    debugf("sigaddset TEST_OK.\n");
    sigdelset(&set,5);//删除id为5的信号
    if(sigismember(&set,3)==0){ //检查3是否还在
        user_panic("wrong in sigdelset_1\n");
    }
    if(sigismember(&set,5)==1){ //检查5是否已经被删除
        user_panic("wrong in sigdelset_2\n");
    }
    debugf("sigdelset TEST_OK.\n");
    sig.sa_handler=handler;
    sig.sa_mask=set;
    if(sigaction(TEST_NUM, &sig, NULL)<0){
        user_panic("first register failed.\n");
```

```
    }
    sig.sa_handler=handler1; //修改处理函数
    if(sigaction(TEST_NUM, &sig, &swap_sig)<0){
        user_panic("second register failed.\n");
    }
    //测试函数是否重新设置
    sig.sa_handler(TEST_NUM);
    if(swap_sig.sa_mask.sig[0]==4){ //0b100,第三位信号
        debugf("register successfully!\n");
    }
    else{
        debugf("register failed.\n");
    }
    //测试交换出来的函数是否正确
    swap_sig.sa_handler(TEST_NUM);
    return 0;
}
```

用于测试信号注册是否正确，以及相关的信号掩码处理函数的正确性。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
sigaddset TEST_OK.
sigdelset TEST_OK.
You have changed register func for sig10.
register successfully!
You have passed register Test for sig10!
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    8001719c  sp:  803ffe80  Status: 0000100c
Cause: 00000020  EPC: 00401970  BadVA:  0040252c
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_2

```
#include <lib.h>
#define TEST_NUM 10
/*
期望输出
You have passed register Test for sig5!
You have passed register Test for sig17!
You have passed register Test for sig23!
You have passed register Test for sig35!
You have passed register Test for sig24!
You have passed register Test for sig56!
*/
void handler(int num){
    debugf("You have passed register Test for sig%d!\n",num);
}
int main(){
```

```
    sigset_t set;
    sigemptyset(&set);
    struct sigaction sig;
    sig.sa_handler=handler;
    sig.sa_mask=set;
    sigaction(5, &sig, NULL);
    sigaction(17, &sig, NULL);
    sigaction(23, &sig, NULL);
    sigaction(35, &sig, NULL);
    sigaction(24, &sig, NULL);
    sigaction(56, &sig, NULL);
    if(kill(0,5)<0){
        user_panic("wrong in kill_1\n");
    }
    if(kill(0,17)<0){
        user_panic("wrong in kill_2\n");
    }
    if(kill(0,23)<0){
        user_panic("wrong in kill_3\n");
    }
    if(kill(0,35)<0){
        user_panic("wrong in kill_4\n");
    }
    if(kill(0,24)<0){
        user_panic("wrong in kill_5\n");
    }
    if(kill(0,56)<0){
        user_panic("wrong in kill_6\n");
    }
    if(kill(0,100)!=-1){ //测试错误条件
        user_panic("wrong in kill_7\n");
    }
    //getwaitlist(0);
    int ans = 0;
    for (int i = 0; i < 10000000; i++) {
        ans += i;
    }
    return 0;
}
```

用于测试进程给自己发送信号的处理。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
You have passed register Test for sig5!
You have passed register Test for sig17!
You have passed register Test for sig23!
You have passed register Test for sig35!
You have passed register Test for sig24!
You have passed register Test for sig56!
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    8001719c  sp:  803ffe80  Status: 0000100c
Cause: 00000020  EPC: 00401950  BadVA:  0040250c
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_3

```c
#include <lib.h>
/*
期望输出
Father: 1.
I can see envid 4097.
Child: 0.
I can see envid 0.
*/
sigset_t set2;
void handler(int num){
    debugf("I can see envid %d.\n",num);
}
int main(int argc, char **argv) {
    sigset_t set;
    struct sigaction sig;
    struct sigaction swap_sig;
    sig.sa_handler=handler;
    sigemptyset(&set);
    sigaddset(&set, 1);
    sigaddset(&set, 2);
    panic_on(sigprocmask(0, &set, NULL));
    sigdelset(&set, 2);
    panic_on(sigaction(10, &sig, NULL));//注册函数
    int ret = fork();
    if (ret != 0) {
        panic_on(sigprocmask(0, &set2, &set));
        debugf("Father: %d.\n", sigismember(&set, 2));
        panic_on(sigaction(10, &sig, &swap_sig));
        swap_sig.sa_handler(ret); //用于检查子进程是否继承了父进程的处理函数
    } else {
        debugf("Child: %d.\n", sigismember(&set, 2));
        panic_on(sigaction(10, &sig, &swap_sig));
        swap_sig.sa_handler(ret); //用于检查子进程是否继承了父进程的处理函数
    }
    return 0;
}
```

用于测试子进程是否继承了父进程的signal_caller以及相应的信号处理函数。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
Father: 1.
I can see envid 4097.
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
Child: 0.
I can see envid 0.
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800172ac  sp:  803ffe80  Status: 0008100c
Cause: 00000020  EPC: 00401960  BadVA:  00403334
curenv:    NULL
cur_pgdir: 83ff3000
```

Lab4_challenge_4

```c
#include <lib.h>
/*
期望输出
Reach handler, now the signum is 3!
Reach handler, now the signum is 2!
global = 2.
*/
int global = 0;
void handler(int num) {
    debugf("Reach handler, now the signum is %d!\n", num);
    global++;
}

#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));
    panic_on(sigaction(3, &sig, NULL));
    sigaddset(&set, TEST_NUM);
    panic_on(sigprocmask(0, &set, NULL));
    kill(0, TEST_NUM);
    kill(0, 3);
    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;
}
```

用于测试进程给自己发送信号的处理。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
Reach handler, now the signum is 2!
Reach handler, now the signum is 3!
global = 2.
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800172ac  sp:  803ffe80  Status: 0000100c
Cause: 00000020  EPC: 00401900  BadVA:  004024bc
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_5

```
#include <lib.h>
/*
Segment fault appear!
test = 1.
*/
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;
}
```

用于测试空指针，为其注册处理函数。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
Segment fault appear!
test = 1.
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800172ac  sp:  803ffe80  Status: 0000100c
Cause: 00001020  EPC: 004017c0  BadVA:  00404000
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_6

```c
#include <lib.h>
/*
期望输出
The process ended successfully.
*/
#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) { //没有注册处理函数，以默认方式处理
    sigset_t set;
    sigemptyset(&set);
    struct sigaction sig;
    sig.sa_handler = sgv_handler;
    sig.sa_mask = set;
    *test = 10;
    debugf("test = %d.\n", *test);
    return 0;
}
```

用于空指针测试，但是不注册处理函数，检查默认情况。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
The process ended successfully.
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800172ac  sp:  803ffe80  Status: 0000100c
Cause: 00001020  EPC: 00401770  BadVA:  0040232c
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_7

```c
#include <lib.h>
/*
标准输出
i am 1001
@@@@@send 3 from 1001 to 800
i am 800
@@@@@send 5 from 800 to 1001
800 received sig2.
1001 received sig5.
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
*/
const int PP1 = 0x800, PP2 = 0x1001;
void handler(int num){
    int envid=syscall_getenvid();
    debugf("%x received sig%d.\n",envid,num);
}
int main() {
  u_int me, who, i;
  me = syscall_getenvid();
  struct sigaction sig;
  sigset_t set;
  sigemptyset(&set);
  sig.sa_handler=handler;
  sig.sa_mask=set;
  sigaction(2,&sig,NULL);
  sigaction(3,&sig,NULL);
  sigaction(5,&sig,NULL);
  debugf("i am %x\n", me);
  if (me == PP1) {
    who = PP2;
    kill(who,5);
    debugf("@@@@@send 5 from %x to %x\n", me, who);
  } else if (me == PP2) {
    who = PP1;
    kill(who,3);
```

```
    debugf("@@@@@send 3 from %x to %x\n", me, who);
  } else {
    debugf("unexpected envid %x\n", me);
    syscall_panic("halt");
  }
  for (i=0;i<10000000;) {
    i++;
    if(((i%100)==0)){
    }
    if (i == 10000) {
      kill(PP1,2);
    }
  }
  return 0;
}
```

用于测试不同进程间的信号发送测试。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
i am 1001
@@@@@send 3 from 1001 to 800
i am 800
@@@@@send 5 from 800 to 1001
800 received sig2.
1001 received sig5.
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800172cc  sp:  803ffe80  Status: 0008100c
Cause: 00000020  EPC: 004018a0  BadVA:  7f3fff68
curenv:    NULL
cur_pgdir: 83ff5000
```

Lab4_challenge_8

```
#include <lib.h>
void handler(int num){
    debugf("I can see envid %d.\n",num);
}
sigset_t set2;
int main(int argc, char **argv) {
    sigset_t set;
    struct sigaction sig;
    struct sigaction swap_sig;
    sigemptyset(&set);
    sigaddset(&set, 1);
    sigaddset(&set, 2);
    sig.sa_handler=handler;
```

```
    sig.sa_mask=set;
    panic_on(sigprocmask(0, &set, NULL));
    sigdelset(&set, 2);
    panic_on(sigaction(10, &sig, NULL));//注册函数
    panic_on(sigaction(5, &sig, NULL));//注册函数
    int ret = fork();
    if (ret != 0) {
        kill(ret,10);
        panic_on(sigprocmask(0, &set2, &set));
        debugf("Father: %d.\n", sigismember(&set, 2));
    } else {
        debugf("Child: %d.\n", sigismember(&set, 2));
    }
    for(int i=0;i<100000;i++);
    return 0;
}
```

用于父子进程之间的信号发送，以及进程掩码设置测试。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
Father: 1.
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
I can see envid 10.
Child: 0.
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800172ac  sp:  803ffe80  Status: 0008100c
Cause: 00000020  EPC: 00401940  BadVA:  00404010
curenv:    NULL
cur_pgdir: 83ff3000
```

Lab4_challenge_9

```
#include <lib.h>
    struct sigset_t set2;
    void handler(int num){
            debugf("HANDLER:%x %d\n",syscall_getenvid(),num);
      }
     int main(int argc, char **argv) {
         struct sigset_t set;
       set.sig[0]=set.sig[1]=0;
        struct sigaction act;
       act.sa_mask=set;
       act.sa_handler=handler;
       sigaction(1,&act,NULL);
       int ret = fork();
       if (ret != 0) {
           kill(ret,1);
```

```
            debugf("Father!!!!!!!!!!\n");
        } else {
            debugf("siojdioqjd\n");
           //while(1);
          }
       return 0;
}
```

用于父子进程之间的信号发送。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
Father!!!!!!!!!!
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
HANDLER:1001 1
siojdioqjd
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800172ac  sp:  803ffe80  Status: 0008100c
Cause: 00000020  EPC: 00401790  BadVA:  00404010
curenv:    NULL
cur_pgdir: 83ff3000
```

Lab4_challenge_10

```c
#include<lib.h>
int global=0;
void handler(int num){
    debugf("Reach handler ,now the signum is %d!\n",num);
    global=1;
}

void handlertwo(int num) {
    debugf("Reach handlertwo ,now the signum is %d!\n",num);
}

struct sigset_t set2;
//测试了子进程会继承父进程的处理函数/写时复制
int main(int argc,char**argv){
    struct sigset_t set;
    struct sigaction sig;
    sigemptyset(&set);
    sigaddset(&set,1);
    sigaddset(&set,2);
    sig.sa_handler=handler;
    sig.sa_mask=set;
    panic_on(sigaction(2,&sig,NULL));
    panic_on(sigprocmask(1,&set,NULL));
    sigdelset(&set,2);
```

```
    int ret=fork(),temp=0;
    if(ret!=0){
        debugf("child is %d\n", ret);
        kill(ret,2);
        for(int i=0;i<1000000;i++)temp*=i;
        debugf("Father : global %d.\n",global);
        }else{
            kill(2048, 2);
            for(int i=0;i<10000;i++)
                temp+=i;
            debugf("Child : global %d.\n",global);
        }
    return 0;
}
```

用于测试父子进程之间的继承关系和写时赋值。

输出如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
child is 4097
Reach handler ,now the signum is 2!
Child : global 1.
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
Reach handler ,now the signum is 2!
Father : global 1.
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    8001723c  sp:  803ffe80  Status: 0008100c
Cause: 00000020  EPC: 00401970  BadVA:  00404000
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_11

```
#include<lib.h>
int global=0;
void handler(int num){
    debugf("Reach handler ,now the signum is %d!\n",num);
    global=1;
}


struct sigset_t set2;
//测试了子进程会继承父进程的处理函数/写时复制
//父进程15号被阻塞，子进程不会阻塞,阻塞SIGTERM号信号，尝试给父进程发送
int main(int argc,char**argv){
    struct sigset_t set;
    struct sigaction sig;
    sigemptyset(&set);
```

```
    sigaddset(&set,1);
    sigaddset(&set,2);
    sigaddset(&set,15);
    sig.sa_handler=handler;
    sig.sa_mask=set;
    panic_on(sigaction(2,&sig,NULL));
    panic_on(sigprocmask(2,&set,NULL));
    int ret=fork(),temp=0;
    if(ret!=0){
        debugf("child is %x\n", ret);
        kill(ret,2);
        for(int i=0;i<1000000;i++)temp+=i;
        debugf("Father : global %d.\n",global);
    }else{
            kill(2048, 2); //还是会被阻塞
            // kill(0,2);
            for(int i=0;i<10000;i++)temp+=i;
            debugf("Child : global %d.\n",global);
            kill(2048,15);
    }
    return 0;
}
```

用于测试掩码，父进程会被阻塞，子进程正常执行。

结果如下:

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
child is 1001
Reach handler ,now the signum is 2!
Child : global 1.
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
Father : global 0.
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800172ac  sp:  803ffe80  Status: 0008100c
Cause: 00000020  EPC: 00401930  BadVA:  00404000
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_12

```
#include <lib.h>
struct sigset_t set2;
    int  cnt=0;
    void handler(int num){
            cnt++;
            debugf("cnt:%d HANDLER:%x %d\n",cnt,syscall_getenvid(),num);
            if(cnt==10){
                  debugf("CONGRATULATION:TEST PASSED!\n");
```

```
                kill(0,SIGKILL);
            }
        }
    int main(int argc, char **argv) {
            struct sigset_t set;
      set.sig[0]=set.sig[1]=0;
       struct sigaction act;
      act.sa_mask=set;
      act.sa_handler=handler;
      sigaction(10,&act,NULL);
    int ret = fork();
      debugf("father:%d child:%d\n",syscall_getenvid(),ret);
      if (ret != 0) {
              for(int i=0;i<10;i++){
                 kill(ret,10);
              }
     } else {
          while(1);
     }
    return 0;
}
```

更强的测试进程间信号发送。

结果如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
father:2048 child:4097
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
cnt:1 HANDLER:1001 10
father:4097 child:0
cnt:2 HANDLER:1001 10
cnt:3 HANDLER:1001 10
cnt:4 HANDLER:1001 10
cnt:5 HANDLER:1001 10
cnt:6 HANDLER:1001 10
cnt:7 HANDLER:1001 10
cnt:8 HANDLER:1001 10
cnt:9 HANDLER:1001 10
cnt:10 HANDLER:1001 10
CONGRATULATION:TEST PASSED!
The process ended successfully.
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    8001723c  sp:  803ffe80  Status: 0008100c
Cause: 00001020  EPC: 00401830  BadVA:  7f3fdf98
curenv:    NULL
cur_pgdir: 83ff3000
```

Lab4_challenge_13

```
#include <lib.h>
void handler1(int num){
    kill(0,2);
    debugf("handler1 arrive!\n");
}
void handler2(int num){
    kill(0,3);
    debugf("handler2 arrive!\n");
}
void handler3(int num){
    debugf("handler3 arrive!\n");
}
int main(){
   struct sigset_t a;
   a.sig[1]=0;
   a.sig[0]=0;
   struct sigaction act;
   act.sa_mask=a;
   act.sa_handler=handler1;
   sigaction(1,&act,NULL);
   act.sa_handler=handler2;
   a.sig[0]|=(1<<2);  //block signal 3
   act.sa_mask=a;
   sigaction(2,&act,NULL);
   act.sa_mask.sig[0]=0;
   act.sa_handler=handler3;
   sigaction(3,&act,NULL);
   kill(0,1);
   for(int i=0;i<10000000;i++);
   for(int i=0;i<10000000;i++);
}
```

用于多处理函数测试。

结果如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
handler2 arrive!
handler1 arrive!
handler3 arrive!
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    8001723c  sp:  803ffe80  Status: 0000100c
Cause: 00000020  EPC: 004018a0  BadVA:  0040245c
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_14

```
#include <lib.h>
struct sigset_t set2;
struct sigset_t set;
int mark = 0;

void handler_5(int num) {
kill(0,10);
debugf("\nsignal 5 should appear secondly mark = %d\n\n",mark);
mark++;
}
void handler_10(int num) {
debugf("\nsignal 10 should appear first mark = %d\n\n",mark);
mark++;
}
void handler_1(int num) {
debugf("\nsignal 1 should appear  mark = %d\n\n",mark);
mark++;
}

int main(int argc, char **argv) {
    sigemptyset(&set);
    struct sigaction sig;
    sig.sa_handler = handler_5;
    sig.sa_mask = set;
    sigaction(5,&sig,NULL);
    struct sigaction sig2;
    sig2.sa_handler = handler_10;
    sig2.sa_mask = set;

    sigaddset(&set, 1);
    sigaddset(&set, 10);
    struct sigaction sig3;
    sig3.sa_handler = handler_1;
    sig3.sa_mask = set;

    sigaction(1,&sig3,NULL);
    sigaction(5,&sig,NULL);
    sigaction(10,&sig2,NULL);
    panic_on(sigprocmask(0, &set, NULL));

    kill(0,1);
    kill(0,5);
    for(int i=0;i<10000000;i++);
    debugf("clear mask\n");
    sigemptyset(&set);
    panic_on(sigprocmask(2, &set, NULL));
    for(int i=0;i<10000000;i++);
    return 0;
 }
```

用于测试信号处理的顺序和重入。

结果如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success

signal 10 should appear first mark = 0


signal 5 should appear secondly mark = 1

clear mask

signal 1 should appear  mark = 2

[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800171bc  sp:  803ffe80  Status: 0000100c
Cause: 00000020  EPC: 00401b40  BadVA:  00402000
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_15

```c
#include<lib.h>
int global=0;
void handler(int num){
    debugf("Reach handler ,now the signum is %d!\n",num);
    global=1;
}

struct sigset_t set2;
int main(int argc,char**argv){
    struct sigset_t set;
    struct sigaction sig;
    sigemptyset(&set);
    sigaddset(&set,1);
    //sigaddset(&set,2);
    sigaddset(&set,15);
    sig.sa_handler=handler;
    sig.sa_mask=set;
    panic_on(sigaction(2,&sig,NULL));
    int ret=fork(),temp=0;
    if(ret!=0){
        debugf("father envid %d\n",syscall_getenvid());
        debugf("child is %d\n", ret);
        if (syscall_getenvid() == 4097) {
            kill(8195, 2);
        } else if (syscall_getenvid() == 2048) {
            kill(6146, 2);
        }
        for(int i=0;i<10000000;i++)temp+=i;
        debugf("Father : global %d.\n",global);
    }else{
            debugf("child envid %d\n",syscall_getenvid());
            if (syscall_getenvid() == 8195) {
```

```
            kill(4097, 2);
        } else if (syscall_getenvid() == 6146) {
            kill(2048, 2);
        }
            for(int i=0;i<10000;i++)temp+=i;
            debugf("Child : global %d.\n",global);
    }
    return 0;
}
```

更强的进程间信号发送测试，四个进程间相互发送信息。

结果如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
father envid 4097
child is 6146
father envid 2048
child is 8195
Reach handler ,now the signum is 2!
child envid 6146
Child : global 1.
[00001802] destroying 00001802
[00001802] free env 00001802
i am killed ...
Reach handler ,now the signum is 2!
child envid 8195
Child : global 1.
[00002003] destroying 00002003
[00002003] free env 00002003
i am killed ...
Reach handler ,now the signum is 2!
Reach handler ,now the signum is 2!
Father : global 1.
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
Father : global 1.
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    8001725c  sp:  803ffe80  Status: 0008100c
Cause: 00000020  EPC: 00401990  BadVA:  00404000
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_16

```
#include <lib.h>
const int PP1 = 0x800, PP2 = 0x1001;
int finished = 0;
void handler(int num){
    debugf("into sighandler to %d.\n",num);
```

```
    int envid=syscall_getenvid();
    debugf("%x received sig%d.\n",envid,num);
    if(num == 5){
      finished++;
      debugf("adding finished!\n");
    }
}
int main() {
  u_int me, who, i=0;
  me = syscall_getenvid();
  struct sigaction sig;
  struct sigset_t set;
  struct sigset_t shed;
  sigemptyset(&set);
  sigemptyset(&shed);
  sigaddset(&shed, 2); //首先屏蔽2/3号
  sigaddset(&shed, 3);
  sig.sa_handler=handler;
  sig.sa_mask=set;
  sigaction(2,&sig,NULL); //2号被屏蔽，不应该优先处理。
  sigaction(3,&sig,NULL);//3号被屏蔽，不应该优先处理。
  sigaction(5,&sig,NULL);//5号设置finished加一
  //由于屏蔽位被替换，PP2应当能收到5、2、3
  debugf("i am %x\n", me);
  sigprocmask(2, &shed, NULL);
  if (me == PP1) {
    kill(PP2, 5);
    debugf("@@@@PP1 sent 5 to %x@@@@\n", PP2);
    debugf("PP1 start wait finished sig\n");
    while(finished <1){
      i++;
      if((i%10000000)  == 0){
        debugf("%x now finished is %d\n", me,finished);
      }
    }
    debugf("pp1received start sig and send!\n");
    who = PP2;
    kill(who, 5);
    kill(who, 3);
    kill(who, 2);
    kill(who, 5);
    kill(who, 5);// 共发送4次5信号，PP2应当均可接收到
    debugf("@@@@PP1 sent 5,3,2 to PP2@@@@\n");
  }if(me == PP2){
    debugf("wait PP1 start!\n");
    while(finished < 1){
      i++;
      if((i%10000000) == 0){
        debugf("%x now finished is %d\n", me,finished);
      }
    }
    //pp2完成初始化，给pp1发送开始信号。
    kill(PP1, 5);
    debugf("@@@@@send 5 from %x to %x\n", me, PP1);
    debugf("PP2 for start\n");
```

```
    while(finished <4){
      i++;
      if((i%10000000) == 0){
        debugf("%x now finished is %d\n", me,finished);
      }
    }
  }
  return 0;
}
```

用于测试信号的重入处理

结果如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
i am 1001
wait PP1 start!
i am 800
@@@@PP1 sent 5 to 1001@@@@
PP1 start wait finished sig
into sighandler to 5.
1001 received sig5.
adding finished!
@@@@@send 5 from 1001 to 800
PP2 for start
into sighandler to 5.
800 received sig5.
adding finished!
pp1received start sig and send!
@@@@PP1 sent 5,3,2 to PP2@@@@
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
into sighandler to 5.
1001 received sig5.
adding finished!
into sighandler to 5.
1001 received sig5.
adding finished!
into sighandler to 2.
1001 received sig2.
into sighandler to 3.
1001 received sig3.
into sighandler to 5.
1001 received sig5.
adding finished!
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1
```

Lab4_challenge_17

```
#include <lib.h>

int global = 0;
void func(int num) {
    global++;
    debugf("Reach handler! Glboal = %d\n", global);
    if (global < 9) {
        kill(0, 4);
    }
    debugf("Leave handler\n");
}
```

```
int main() {
    debugf("Recursion test\n");
    struct sigaction mysigaction;
    mysigaction.sa_handler = func;
    sigemptyset(&mysigaction.sa_mask);
    sigaction(4, &mysigaction, NULL);
    kill(0, 4);
    return 0;
}
```

用于测试信号重入

结果如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
Recursion test
Reach handler! Glboal = 1
Reach handler! Glboal = 2
Reach handler! Glboal = 3
Reach handler! Glboal = 4
Reach handler! Glboal = 5
Reach handler! Glboal = 6
Reach handler! Glboal = 7
Reach handler! Glboal = 8
Reach handler! Glboal = 9
Leave handler
Leave handler
Leave handler
Leave handler
Leave handler
Leave handler
Leave handler
Leave handler
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800171bc  sp:  803ffe80  Status: 0000100c
Cause: 00000020  EPC: 004018a0  BadVA:  7f3fff68
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_18

```
#include <lib.h>
int cnt=0;
void handler(int num){
    cnt++;
    if(cnt>5){
        return;
    }
    debugf("cnt:%d HANDLER:%x %d\n",cnt,syscall_getenvid(),num);
}

int main(int argc, char **argv) {
```

```
//先注册了一个信号
    struct sigaction act;
    sigemptyset(&act.sa_mask);
    act.sa_handler=handler;
    sigaction(10,&act,NULL);
    int father = syscall_getenvid();
    int ret = fork();
    if (ret != 0) {
        struct sigset_t set;
            sigemptyset(&set);
            for(int i=0;i<5;i++){
            kill(ret,10);
        }
        while(cnt!=5);
        debugf("Father passed!\n");
    } else {
    while(cnt!=5);
    debugf("Child passed!\n");
    for(int i=0;i<5;i++){
        kill(father,10);
    }
    }
return 0;
}
```

用于测试父子进程发送信号以及结束

结果如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
cnt:1 HANDLER:1001 10
cnt:2 HANDLER:1001 10
cnt:3 HANDLER:1001 10
cnt:4 HANDLER:1001 10
cnt:5 HANDLER:1001 10
Child passed!
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
cnt:1 HANDLER:800 10
cnt:2 HANDLER:800 10
cnt:3 HANDLER:800 10
cnt:4 HANDLER:800 10
cnt:5 HANDLER:800 10
Father passed!
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800171bc  sp:  803ffe80  Status: 0008100c
Cause: 00000020  EPC: 004019a0  BadVA:  00403290
curenv:    NULL
cur_pgdir: 83ffc000
```

Lab4_challenge_19

```
#include <lib.h>
const int PP1 = 0x800, PP2 = 0x1001;
void handler(int num){
    debugf("into sighandler to %d.\n",num);
    int envid=syscall_getenvid();
```

```
    debugf("%x received sig%d.\n",envid,num);
}
int main() {
  u_int me, who, i=0;
  me = syscall_getenvid();
  struct sigaction sig;
  struct sigset_t set;
  struct sigset_t shed;
  sigemptyset(&set);
  sigemptyset(&shed);
  sigaddset(&shed, 2); //首先屏蔽2/3号
  sigaddset(&shed, 3);
  sig.sa_handler=handler;
  sig.sa_mask=set;
  sigaction(2,&sig,NULL); //2号被屏蔽，不应该优先处理。
  sigaction(3,&sig,NULL);//3号被屏蔽，不应该优先处理。
  sigaction(5,&sig,NULL);
  debugf("i am %x\n", me);
  sigprocmask(2, &shed, NULL);
  if (me == PP1) {
    who = PP2;
    kill(who, 2);
    kill(who, 3);
    kill(who, 5);
    debugf("@@@@PP1 sent 2,3,5 to PP2@@@@\n");
  }if(me == PP2){
    for(int i=0;i<1000000;i++);
  }
  return 0;
}
```

用于测试信号的屏蔽以及处理顺序

结果如下：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
i am 1001
i am 800
@@@@PP1 sent 2,3,5 to PP2@@@@
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
into sighandler to 5.
1001 received sig5.
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
panic at sched.c:50 (schedule): error in schedule 1

ra:    800171dc  sp:  803ffe80  Status: 0008100c
Cause: 00000020  EPC: 004019b0  BadVA:  7f3fff68
curenv:    NULL
cur_pgdir: 83ff5000
```