Lab4-Challenge答辩

北航航空航天大学/计算机学院/操作系统课程挑战性任务

汇报人：210615小班 张杨

Blog： yanna-zy.gitee.io

目录

Contents

壹

任务实现思路

Task Implementation Ideas

任务实现思路-信号变量的定义

（一）信号变量的定义

- 在**include文件夹**中**新建**一个文件**signal.h**，其中定义了struct sigset_t、struct sigaction、struct sigstack 结构体，其中 **struct sigstack** 是用来**存储进程接收到的信号的链表节点**
- 在include/env.h的结构体 **struct Env** 中加入有关信号处理的属性：

```
struct Env {
    //......

    //Lab4 challenge
    u_int env_handlers[65];
    sigset_t env_sa_mask;
    struct sigstack *env_sig_stack;
    u_int env_sig_entry;
    u_int env_sig_flag;
    u_int env_protect[256];
};
```

任务实现思路-信号函数的定义与实现

（二）信号函数的定义与实现

在**user/include/lib.h**定义信号相关的**用户处理函数**，并在**user/lib/signal.c**中**实现**上述函数。

- **第一类：简单的信号函数**

  五个用来处理信号掩码的函数。

- **第二类：需要系统调用的信号函数**

  函数 sigaction、sigprocmask、kill 由于需要**改变进程控制块中的信号属性**，所以需要利用**系统调用**陷入内核态进行处理。

```
int syscall_kill(u_int envid, int sig);
int syscall_set_sig_act(u_int envid, int signum, const struct sigaction *act);
int syscall_set_sig_set(u_int envid, int how, const sigset_t *set, sigset_t *oldset);
int syscall_get_sig_act(u_int envid, int signum, struct sigaction *oldact);
```

任务实现思路-用户态异常处理函数

（三）用户态异常处理函数

1. 异常处理函数的定义

在user/lib/fork.c文件中，实现异常处理函数 sig_entry：

```
static void __attribute__((noreturn)) sig_entry(struct Trapframe *tf,
                                                void (*sa_handler)(int), int signum, int envid) {
    if (sa_handler != 0) {
        sa_handler(signum); //直接调用定义好的处理函数
        int r = syscall_set_sig_trapframe(0, tf);
        user_panic("sig_entry syscall_set_trapframe returned %d", r);
    }
    switch (signum) {
        case SIGKILL: case SIGSEGV: case SIGTERM:
            syscall_env_destroy(envid); //默认处理
            user_panic("sig_entry syscall_env_destroy returned");
        default:;
            int r = syscall_set_sig_trapframe(0, tf);
            user_panic("sig_entry syscall_set_trapframe returned %d", r);
    }
}
```

syscall_set_sig_trapframe 函数仿照 syscall_set_trapframe 实现，但需要注意将进程的**信号处理标志位 env_sig_flag 置零**。

任务实现思路-用户态异常处理函数

（三）用户态异常处理函数

2. 异常处理函数的跳转

在kern/genex.S的**汇编**函数 **ret_from_exception** 中加入到**跳转**到 do_signal 的代码：

```
FEXPORT(ret_from_exception)
    /*-----------------adding codes--------------*/
    move    a0, sp
    addiu   sp, sp, -8
    jal        do_signal
    nop
    addiu   sp, sp, 8
    /*----------------finish codes---------------*/
    RESTORE_SOME
    lw      k0, TF_EPC(sp)
    lw      sp, TF_REG29(sp) /* Deallocate stack */
.set noreorder
    jr      k0
    rfe
.set reorder
```

do_signal 函数的功能是：判断并**选定**接下里需要处理的**信号**，并设置好接下来要跳转到 sig_entry 用户态处理函数，即将 **tf->cp0_epc** 改成 **sig_entry** 的入口地址。【仿照：do_tlb_mod】

任务实现思路-其他步骤

（四）其他步骤

子进程继承父进程的信号处理函数

- 将父进程有关信号的用户态异常处理函数继承给子进程，具体操作就是将 **sig_entry 函数**的地址传给子进程 env_sig_entry 变量。
- 将父进程的**信号处理函数**继承给子进程，注意64个处理函数都要继承。

信号默认处理动作设置

对于 SIGSEGV 信号，通过观察我发现 **UTEMP = 0x3FE000**，于是我将kern/tlbex.c里面的 **passive_alloc** 函数中对 va < UTEMP 的处理修改成**发送11号信号**：

```
static void passive_alloc(u_int va, Pde *pgdir, u_int asid) {
    //......
    /* 判断地址的合法性*/
    if (va < UTEMP) {
        sys_kill(0, SIGSEGV);
        //panic("address too low");
    }
    //......
}
```

贰

测试程序结果

Test Program Results

测试程序结果-课程组数据

基本信号测试

空指针测试

写时复制测试

```
git@21373002:~/21373002 (lab4-challenge)$ make test
gxemul -T -C R3000 -M 64 -E testmips -d target/fs.i
GXemul 0.7.0+dfsg    Copyright (C) 2003-2021  Ander
Read the source code and/or documentation for other

net:
    simulated network:
        10.0.0.0/8 (max outgoing: TCP=100, UDP=100)
        gateway+nameserver: 10.0.0.254 (60:50:40:36
        nameserver uses real nameserver 202.112.128
machine:
    model: MIPS test machine
    cpu: R3000 (32-bit, I+D = 4+4 KB, L2 = 0 KB)
    memory: 64 MB
    diskimage: target/fs.img
        IDE DISK id 0, read/write, 4 MB (9072 512-b
    file: loading target/mos
    cpu0: starting at 0x80011900 <_start>
------------------------------------------------

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!
global = 1.
```

```
git@21373002:~/21373002 (lab4-challenge)$ make test l
gxemul -T -C R3000 -M 64 -E testmips -d target/fs.img
GXemul 0.7.0+dfsg    Copyright (C) 2003-2021  Anders
Read the source code and/or documentation for other C

net:
    simulated network:
        10.0.0.0/8 (max outgoing: TCP=100, UDP=100)
        gateway+nameserver: 10.0.0.254 (60:50:40:30:2
        nameserver uses real nameserver 202.112.128.5
machine:
    model: MIPS test machine
    cpu: R3000 (32-bit, I+D = 4+4 KB, L2 = 0 KB)
    memory: 64 MB
    diskimage: target/fs.img
        IDE DISK id 0, read/write, 4 MB (9072 512-byt
    file: loading target/mos
    cpu0: starting at 0x80011900 <_start>
------------------------------------------------

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.
```

```
git@21373002:~/21373002 (lab4-challenge)$ make test lab=6_1>NUL && make ru
gxemul -T -C R3000 -M 64 -E testmips -d target/fs.img target/mos
GXemul 0.7.0+dfsg    Copyright (C) 2003-2021  Anders Gavare
Read the source code and/or documentation for other Copyright messages.

net:
    simulated network:
        10.0.0.0/8 (max outgoing: TCP=100, UDP=100)
        gateway+nameserver: 10.0.0.254 (60:50:40:30:20:10)
        nameserver uses real nameserver 202.112.128.51
machine:
    model: MIPS test machine
    cpu: R3000 (32-bit, I+D = 4+4 KB, L2 = 0 KB)
    memory: 64 MB
    diskimage: target/fs.img
        IDE DISK id 0, read/write, 4 MB (9072 512-byte blocks)
    file: loading target/mos
    cpu0: starting at 0x80011900 <_start>
------------------------------------------------------------------------

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 ...
Child: 0.
```

测试程序结果-其他数据

递归发出信号测试

```
void handler(int num){
    cnt++;
    if(cnt == 1) {
        kill(0,5);
    }
    debugf("cnt:%d HAN
}
```

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 163
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
cnt:2 HANDLER:1001 5
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
cnt:11 HANDLER:1001 10
father:4097 child:0
[00001001] destroying 00001001
[00001001] free env 00001001
i am killed ...
```

防爆栈测试

```
int father=syscall_getenvid();
int ret = fork();
debugf("father:%d child:%d\n",syscall_gete
if (ret != 0) {
    for(int i=0;i<10;i++) {
        kill(ret,10);
    }
    while(cnt!=10);
    debugf("Father passed!\n");
} else {
    for(int i=0;i<10;i++) {
        kill(father,10);
    }
    while(cnt!=10);
    debugf("Child passed!\n");
}
```

```
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
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
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
father:4097 child:0
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
cnt:6 HANDLER:800 10
cnt:7 HANDLER:800 10
cnt:8 HANDLER:800 10
cnt:9 HANDLER:800 10
cnt:10 HANDLER:800 10
Father passed!
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
```

叁

难点细节实现

Difficult Details Implementation

难点细节实现-sigset_t结构体的定义

1、sigset_t结构体的定义

通过观察题干给出的几个与信号处理相关函数，我发现函数**使用 sigset_t 来表示 struct sigset_t**，因此在signal.h文件中定义 struct sigset_t 结构体时需要使用 **typedef** 的语法：

```
typedef struct sigset_t{
    int sig[2]; //最多 32*2=64 种信号
    //sig[0] 表示1~32  sig[1] 表示33~64
} sigset_t;
```

难点细节实现-头文件的单次展开

2、头文件的单次展开

我原本在处理上述 sigset_t 结构体定义问题中，是使用的 #define 宏来实现，但是出现了一个问题是：当第二次处理signal.h文件的时候，会导致实际 sigset_t **递归两次展开**而出错。

所以，后来我观察了一下include文件夹中的其他.h文件，发现我们需要在每个.h文件中加入如下的开头和结尾，实现**单次展开**：

```
#ifndef _SIGNAL_H_
#define _SIGNAL_H_
//......
#endif /* _SIGNAL_H_ */
```

难点细节实现-用静态数组申请信号栈空间

3、用静态数组申请信号栈空间

由于**局部变量**会在函数结束之后释放，所以在处理 e->env_sig_stack 的压栈记录数据时，我模仿了env_alloc 函数向 envs[] 数组申请进程控制块的机制，定义了一个**静态数组 SIGstack[4096] 用来申请栈空间**：

```
static struct sigstack SIGstack[4096];

int sys_kill(u_int envid, int sig) {
    //......
    for (int i = 0; i < 4096; i++) {
        if (SIGstack[i].next == NULL) {
            SIGstack[i].sig = sig;
            SIGstack[i].next = env->env_sig_stack;
            env->env_sig_stack = &(SIGstack[i]);
            break;
        }
    }
    //......
}
```

难点细节实现-给进程设置异常处理函数入口地址的预处理

4、给进程设置异常处理函数入口地址的预处理

仿照TLB_mod异常处理函数入口地址是在 fork 函数中，使用系统调用为父子进程设置 cow_entry 异常处理函数入口地址。因此，我选择在 **sigaction 注册函数**中，使用**用户态函数 env_set_sig_entry** 函数为进程设置异常处理函数入口地址：

```
int env_set_sig_entry(void) {
    try(syscall_set_sig_entry(0, sig_entry));
    try(syscall_set_tlb_mod_entry(0, cow_entry));
    return 0;
}
```

【注意】：特别值得注意的是，我们这里为进程设置 sig_entry 函数入口地址的同时，也需要**为进程设置 cow_entry 函数入口地址**，防止在后续写时复制的时候出现 panic("TLB Mod but no user handler registered"); 异常。

难点细节实现-从do_signal函数向sig_entry函数传参

5、从do_signal函数向sig_entry函数传参

如下 **sig_entry** 用户异常处理函数定义了四个参数：

```
typedef void __attribute__((noreturn)) (*sig_entry_t)(struct Trapframe *tf,
                                                       void (*sa_handler)(int), int signum, int envid);
```

在 **do_signal** 中用 tf 进行传参：

```
void do_signal(struct trapframe **tf) {
    //......
    if (curenv->env_sig_entry) {
        tf->regs[4] = tf->regs[29]; //参数1
        tf->regs[5] = (unsigned int)(curenv->env_sig_act.sa_handler); //参数2
        tf->regs[6] = sig_stack->sig; //参数3
        tf->regs[7] = curenv->env_id; //参数4
        tf->regs[29] -= sizeof(tf->regs[4]);
        tf->regs[29] -= sizeof(tf->regs[5]);
        tf->regs[29] -= sizeof(tf->regs[6]);
        tf->regs[29] -= sizeof(tf->regs[7]);

        tf->cp0_epc = curenv->env_sig_entry;
    } else {
        panic("sig but no user handler registered");
    }
```

难点细节实现-预防爆栈处理

6、预防爆栈处理

在 do_signal 中判断是否需要转到用户态异常处理函数的时候，需要提前判断一下是否当前已经有进程在处理 sig_entry，避免**爆掉异常处理栈**。

【注意】：如果是 SIGSEGV = 11 的信号则需要“**允许进行重入优先处理**”。

```
void do_signal(struct Trapframe *tf) {
    //......
    if (curenv->env_sig_flag != 0 && sig_stack->sig != SIGSEGV) {
        return;
    }
    curenv->env_sig_flag = 1;
    //......
}
```

难点细节实现-内存泄露envs数组溢出处理

7、内存泄露envs数组溢出处理

在 struct Env 结构体的最后加上一个**长数组保护**，通过反复试验，我最终定义了256大小的 u_int 数组：

```
struct Env {
    //......
    u_int env_protect[256];
};
```

难点细节实现-编译链接.o文件

8、编译链接.o文件

在最初使用 make run 来运行样例数据的时候，我始终出现编译错误，在后续询问同学之后，才知道需要在user/include.mk中加入**链接的signal.o**文件：

```
USERLIB             := entry.o \
                       syscall_wrap.o \
                       debugf.o \
                       libos.o \
                        fork.o \
                       syscall_lib.o \
                       ipc.o \
                       signal.o
```

THANK YOU

谢谢您的聆听