華中科技大學

操作系统课程设计

院　　系 _____

专业班级 _____

姓　　名 _____

学　　号 _____

指导教师 _____

1926 年 8 月 17 日

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

学位论文属于 1、保密 □，在　　年解密后适用本授权书。

2、不保密 □

请在以上相应方框内打 “√”

作者签名：　　年　　月　　日

导师签名：　　年　　月　　日

目　　录

1 打印异常代码行

1.1 实验目的

修改 pke 内核，使得程序在用户态试图读取内核态寄存器时，抛出错误并且输出触发异常的用户程序源文件名和对应代码行。正确的程序输出如下。

```
riscv-pke on  lab1_challenge2_errorline
❯ spike ./obj/riscv-pke obj/app_errorline
In m_start, hartid:0
HTIF is available!
(Emulated) memory size: 2048 MB
Enter supervisor mode...
Application: obj/app_errorline
Application program entry point (virtual address): 0x0000000081000000
Switch to user mode...
Going to hack the system by running privilege instructions.
Runtime error at user/app_errorline.c:13
  asm volatile("csrw sscratch, 0");
Illegal instruction!
System is shutting down with exit code -1.
```

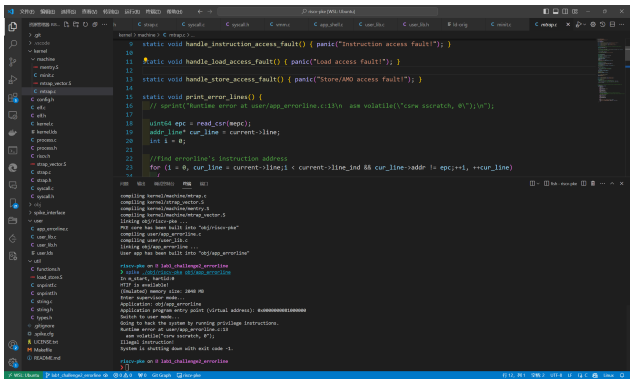

图 1: 正确的输出 lab1_challenge2

1.2 实验内容

为了完成实验，需要执行并且完成以下的两部分内容。

①首先，需要增加对 elf 可执行文件的调试探查。具体来说，就是在 elf.h 头文件中定义，在 elf.c 文件中实现函数 *elf_status load_debugline(elf_ctx *ctx);* ,并且在 elf 的二进制程序加载执行函数 *void load_bincode_from_host_elf(process *p)*中运行以加载 debug 段。

以下是 *load_debugline* 的具体代码以及思路。

```
// added @lab1_challenge2 | load debugline
elf_status load_debugline( elf_ctx *ctx ) {
    // sprint( "hello load_debugline\n");
    // sprint("%d\n",ctx);
    // claim string table header
    elf_sect_header strtab;
    // get string table header's info
    int offset;

    offset = ctx->ehdr.shoff;
    // sprint("%d\n",ctx->ehdr.version);
    // sprint( "%d\n", offset );
    // sprint( "%d\n", ctx->info );
    offset += sizeof( strtab ) * ( ctx->ehdr.shstrndx );
    // sprint( "%d\n", offset );

    // save string table header
   if ( elf_fpread( ctx, (void *) &strtab, sizeof( strtab ), offset ) !
=
         sizeof( strtab ) ) {
        panic( "string table header get failed!\n" );
    }

    // save string table
    static char strtab_info[ STRTAB_MAX ];
       if  (  elf_fpread(  ctx,  (void  *)  strtab_info,  strtab.size,
strtab.offset ) !=
         strtab.size ) {
        panic( "string table get failed!\n" );
    }

    // get .debug_line segment
    elf_sect_header debugseg;
    int index;
   for ( index = 0, offset = ctx->ehdr.shoff; index < ctx->ehdr.shnum;
          index++, offset += sizeof( debugseg ) ) {
                  if   (   elf_fpread(   ctx,   (void   *)   &debugseg,
sizeof( elf_sect_header ),
                         offset ) != sizeof( elf_sect_header ) )
```

```
37              panic( "debug header get failed!\n" );
38          // sprint("i=%d,%s\n",i,strtab_info+debugseg.name);
39          if ( strcmp( (char *) ( strtab_info + debugseg.name ),
40                       ".debug_line" ) == 0 )
41              break;
42      }
43
44      if ( index == ctx->ehdr.shnum ) {
45          panic( "can't find debugline!\n" );
46          return EL_ERR;
47      }
48      int i;
49
50      // get debugline's information
51      static char debugline[ DEBUGLINE_MAX ];
52      if ( elf_fpread( ctx, (void *) debugline, debugseg.size,
53                       debugseg.offset ) != debugseg.size )
54          panic( "debugline get failed!\n" );
55      make_addr_line( ctx, debugline, debugseg.size );
56      return EL_OK;
57  }
```

函数主要思路如下：

1.首先，从 ELF 上下文中获取字符串表头信息。这个信息包含了字符串表的偏移和大小等。

2.然后，通过上一步得到的字符串表头信息，从 ELF 文件中读取字符串表的内容，并保存到数组 *strtab_info* 中。

3.接下来，遍历 ELF 文件中的节头表，寻找名称为.debug_line 的段。

4.一旦找到了.debug_line 段，就从文件中读取该段的内容，保存到另一个数组中。这些内容包含调试信息，比如源代码文件名、行号等。

5.最后，调用 make_addr_line 函数来处理这些调试线信息将缓冲区指针传入。

6.函数返回一个表示加载调试线信息是否成功的状态，EL_OK 表示成功，或者 EL_ERR 表示失败。

②完成对 elf 二进制程序的 debug_line 段的读取以后，接下来要做的就是抛出异常时打印出源程序名和对应代码行。

异常属于中断的一种，因此通过 *mstrap.c* 中定义实现并且在触发 *handle_illegal_instruction* 函数时候执行 *print_error_lines* 函数，就可以正确的完成需求。以下是 *print_error_lines* 函数的具体代码以及实现。

```
static void print_error_lines() {
    //  sprint("Runtime  error  at  user/app_errorline.c:13\n    asm volatile(\"csrw sscratch, 0\");\n");

  uint64 epc = read_csr(mepc);
  addr_line* cur_line = current->line;
  int i = 0;

  //find errorline's instruction address
   for (i = 0, cur_line = current->line;i < current->line_ind && cur_line->addr != epc;++i, ++cur_line)
    {
    //sprint("i=%d,addr=%x\n",i,cur_line->addr);
    if (cur_line->addr == epc) break;
    }
  if (i == current->line_ind) panic("can't find errorline!\n");
  //sprint("find errorline,i=%d\n",i);


//find file's path and name
  char filename[FILENAME_MAX];
  //find filename
  code_file* cur_file = current->file + cur_line->file;
  char* single_name = cur_file->file;
  //find file's path
  char* file_path = (current->dir)[cur_file->dir];
  //combine path and name
  int start = strlen(file_path);
  strcpy(filename, file_path);
  filename[start] = '/';
  start++;
  strcpy(filename + start, single_name);
  sprint("Runtime error at %s:%d\n", filename, cur_line->line);

  //find error line
    //error instruction's line
  int error_line = cur_line->line;
  //open file
  spike_file_t* file = spike_file_open(filename, O_RDONLY, 0);
  if (IS_ERR_VALUE(file)) panic("open file failed!\n");
  //get file's content
  char file_detail[FILE_MAX];
  spike_file_pread(file, (void*)file_detail, sizeof(file_detail), 0);
```

```
42
43    //fine error line's start
44    int line_start = 0;
45    for (i = 1;i < error_line;i++)
46      {
47      //sprint("line=%d,%s\n",i,file_detail+line_start);
48      while (file_detail[line_start] != '\n') line_start++;
49      line_start++;
50      }
51    char* errorline = file_detail + line_start;
52    while (*errorline != '\n') errorline++;
53    *errorline = '\0';
54    //print error line
55    sprint("%s\n", file_detail + line_start);
56    spike_file_close(file);
57    }
```

函数的主要思路如下：

1.寄存器 epc 是用于保存当前执行代码的地址的处理器寄存器，因此为了获得抛出错误的代码位置需要首先获得代码的地址。*read_csr(mepc)*获取发生错误的代码地址。

2.遍历当前进程的代码行数组，在其中查找发生异常的指令地址对应的代码行。

3.根据进程提供信息，拼接获取代码文件的路径和文件名。

4.打开代码文件，读取文件内容。定位到错误行在文件内容中的起始位置。

5.提取错误行的内容并打印出来，同时关闭文件。

1.3 实验调试及心得

关于 elf 文件与进程函数之间的接口问题：读取 elf 文件的代码，找到包含调试信息的段以后，想要将其内容保存起来，有两个可以的手段。要么是用静态数组来存储 debug_line 段数据，那么这个数组必须足够大；也可以把 debug_line 直接放在程序所有需映射的段数据之后，这样可以保证有足够大的动态空间。本次挑战因为提供了方便的 util 函数 *make_addr_line* 来解析.debug_line 段并保存到静态数组，因此选择后者。

2 堆空间管理

2.1 实验目的

2.2 实验内容

2.3 实验调试及心得