ICS 第 16 周小班练习

22 期末第三题 (链接)

本题基于下列 C 语言文件所编译生成的 `main.o` 和 `addvec.o`.
编译和运行在 x86-64/Linux 下完成, 编译过程使用 `-Og` 优化选项.

```c
// main.c
#include <stdio.h>
#include "addvec.h"
int x[2] = {1, 2}, z[2];
int main() {
  int y[2] = {3, 4}, n = 2;
  addvec(x, y, z, n);
  printf("z = [%d %d]\n", z[0], z[1]);
}
```

```c
// addvec.c
#include "addvec.h"

void addvec(int *x, int *y, int *z, int n) {
  for (int i = 0; i < n; i++)
    z[i] = x[i] + y[i];
}
```

对于每个下表中给出的符号, 请用“是”或“否”指出它是否在 `main.o` 的 `.symtab` 节中有符号表条目.
如果存在条目, 则请指出该符号的类型 (局部, 全局或外部), 并指出定义该符号的模块 (`main.o` 或 `addvec.o`),
以及此符号**在该模块所处的节或伪节名**. 如果不存在条目, 则请将该行后继空白处标记为 /.

符号	.symtab 条目?	符号类型	定义符号的模块	节或伪节名
x				
y				
z				
n				
addvec				

基于 `main.o` 和 `addvec.o` 生成可执行程序 `main` 的时候, 会进行若干重定位.
下述代码是 `objdump -D main` 的部分输出, 请补全表格中 `main.o` 的重定位信息. (无需补全汇编代码)

```
Disassembly of section .interp:

0000000000000318 <.interp>:
 318: 2f                   (bad)
 319: 6c                   insb   (%dx),%es:(%rdi)
 31a: 69 62 36 34 2f 6c 64 imul   $0x646c2f34,0x36(%rdx),%esp
 321: 2d 6c 69 6e 75       sub    $0x756e696c,%eax
 326: 78 2d                js     355 <__abi_tag-0x37>
 328: 78 38                js     362 <__abi_tag-0x2a>
 32a: 36 2d 36 34 2e 73    ss sub $0x732e3436,%eax
 330: 6f                   outsl  %ds:(%rsi),(%dx)
 331: 2e 32 00             cs xor (%rax),%al

(----------------------------------------------------------)

Disassembly of section .plt:

0000000000001020 <.plt>:
    1020:   ff 35 92 2f 00 00       push   0x2f92(%rip)
    1026:   ff 25 94 2f 00 00       jmp    *0x2f94(%rip)
    102c:   0f 1f 40 00             nopl   0x0(%rax)
    1030:   f3 0f 1e fa             endbr64
    1034:   68 00 00 00 00          push   $0x0
    1039:   e9 e2 ff ff ff          jmp    1020 <_init+0x20>
    103e:   66 90                   xchg   %ax,%ax
    1040:   f3 0f 1e fa             endbr64
```

```
    1044:   68 01 00 00 00          push   $0x1
    1049:   e9 d2 ff ff ff          jmp    1020 <_init+0x20>
    104e:   66 90                   xchg   %ax,%ax

Disassembly of section .plt.got:

0000000000001050 <__cxa_finalize@plt>:
    1050:   f3 0f 1e fa             endbr64
    1054:   ff 25 9e 2f 00 00       jmp    *0x2f9e(%rip)
    105a:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)

Disassembly of section .plt.sec:

0000000000001060 <__stack_chk_fail@plt>:
    1060:   f3 0f 1e fa             endbr64
    1064:   ff 25 5e 2f 00 00       jmp    *0x2f5e(%rip)
    106a:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)

0000000000001070 <__printf_chk@plt>:
    1070:   f3 0f 1e fa             endbr64
    1074:   ff 25 56 2f 00 00       jmp    *0x2f56(%rip)
    107a:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)

Disassembly of section .text:

(------------------------------------------------------------)

0000000000001169 <main>:
    1169:   f3 0f 1e fa             endbr64
    116d:   48 83 ec 18             sub    $0x18,%rsp
    1171:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    1178:   00 00
    117a:   48 89 44 24 08          mov    %rax,0x8(%rsp)
    117f:   31 c0                   xor    %eax,%eax
    1181:   c7 04 24 03 00 00 00    movl   $0x3,(%rsp)
    1188:   c7 44 24 04 04 00 00    movl   $0x4,0x4(%rsp)
    118f:   00
    1190:   48 89 e6                mov    %rsp,%rsi
    1193:   b9 02 00 00 00          mov    $0x2,%ecx
    1198:   48 8d 15 ____(1)____    lea    ______(%rip),%rdx
    119f:   48 8d 3d ____(2)____    lea    ______(%rip),%rdi
    11a6:   e8 ____(3)____          call   _____________
    11ab:   8b 0d 73 2e 00 00       mov    0x2e73(%rip),%ecx
    11b1:   8b 15 69 2e 00 00       mov    0x2e69(%rip),%edx
    11b7:   48 8d 35 46 0e 00 00    lea    0xe46(%rip),%rsi
    11be:   bf 02 00 00 00          mov    $0x2,%edi
    11c3:   b8 00 00 00 00          mov    $0x0,%eax
    11c8:   e8 a3 fe ff ff          call   1070 <__printf_chk@plt>
    11cd:   48 8b 44 24 08          mov    0x8(%rsp),%rax
    11d2:   64 48 2b 04 25 28 00    sub    %fs:0x28,%rax
    11d9:   00 00
    11db:   75 0a                   jne    11e7 <main+0x7e>
    11dd:   b8 00 00 00 00          mov    $0x0,%eax
    11e2:   48 83 c4 18             add    $0x18,%rsp
    11e6:   c3                      ret
    11e7:   e8 74 fe ff ff          call   1060 <__stack_chk_fail@plt>
```

```
00000000000011ec <addvec>:
    11ec:   f3 0f 1e fa             endbr64
    11f0:   b8 00 00 00 00          mov    $0x0,%eax
    11f5:   eb 12                   jmp    1209 <addvec+0x1d>
    11f7:   4c 63 c0                movslq %eax,%r8
    11fa:   46 8b 0c 86             mov    (%rsi,%r8,4),%r9d
    11fe:   46 03 0c 87             add    (%rdi,%r8,4),%r9d
    1202:   46 89 0c 82             mov    %r9d,(%rdx,%r8,4)
    1206:   83 c0 01                add    $0x1,%eax
    1209:   39 c8                   cmp    %ecx,%eax
    120b:   7c ea                   jl     11f7 <addvec+0xb>
    120d:   c3                      ret

(-----------------------------------------------------------)

Disassembly of section .got:

0000000000003fb0 <_GLOBAL_OFFSET_TABLE_>:
    3fb0:   c0 3d 00 00 00 00 00    sarb   $0x0,0x0(%rip)
    ...
    3fc7:   00 30                   add    %dh,(%rax)
    3fc9:   10 00                   adc    %al,(%rax)
    3fcb:   00 00                   add    %al,(%rax)
    3fcd:   00 00                   add    %al,(%rax)
    3fcf:   00 40 10                add    %al,0x10(%rax)
    ...

Disassembly of section .data:

0000000000004000 <__data_start>:
    ...

0000000000004008 <__dso_handle>:
    4008:   08 40 00                or     %al,0x0(%rax)
    400b:   00 00                   add    %al,(%rax)
    400d:   00 00                   add    %al,(%rax)
    ...

0000000000004010 <x>:
    4010:   01 00                   add    %eax,(%rax)
    4012:   00 00                   add    %al,(%rax)
    4014:   02 00                   add    (%rax),%al
    ...
```

编号	r.offset	r.type	r.symbol	r.addend	填入的值
(1)	0x32				
(2)	0x39				
(3)	0x3e	R_X86_64_PC32			

生成该可执行程序的操作系统的动态链接器的路径是?
提示: 字符 `-./0a` 的 ASCII 值分别为 `0x2d`, `0x2e`, `0x2f`, `0x30`, `0x61`.

程序中 `__printf_chk` 函数的 PLT 表条目是 `PLT[ ]`, GOT 表条目是 `GOT[ ]`. (填写数字即可)

得分

第五题（10 分）

代码如下：（仅为示例代码，答题时不考虑任何出错的情况）

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int counter1 = 1;
int counter2 = 1;

void handler(int sig) {
       if (sig == SIGUSR1)    counter1++;
       if (sig == SIGUSR2)    counter2++;
}

int main(int argc, char **argv)
{
       int i;
       pid_t ppid, cpid;
       int fd1, fd2;
       unsigned char buf1[64], buf2[64];

       signal(SIGUSR1, handler);
       signal(SIGUSR2, handler);

       fd1 = open("number.txt", O_RDWR);

       for (i = 0; i < __N__; i++) {
              ppid = getpid();
              cpid = fork();
              if (cpid) {
```

```
                kill(cpid, ___A_____ );
                while (sleep(4)) ;
        } else {
                kill(ppid, ___B_____ );
                while (sleep(2)) ;
        }
        fd2 = open("letter.txt", O_RDWR);
        read(fd1, buf1, counter1);
        read(fd2, buf2, counter2);
        buf1[counter1] = '\0';
        buf2[counter2] = '\0';
        printf("%s %s\n", buf1, buf2);
        if (cpid) {
                waitpid(cpid, 0, 0);
        } else {
                _____E_____;
        }
    }
}
```

其中，number.txt 文件内容为：

12345678901234567890123456789O

其中，letter.txt 文件内容为：

abcdefghijklmnopqrstuvwxyz

1．当 N=2，A/B 为 SIGUSR1 或 SIGUSR2，E 为空时，共有_____行输出

a） A 为 SIGUSR1，B 为 SIGUSR1 时，最后一行输出为_____

b） A 为 SIGUSR1，B 为 SIGUSR2 时，最后一行输出为_____

c） A 为 SIGUSR2，B 为 SIGUSR1 时，最后一行输出为_____

d） A 为 SIGUSR2，B 为 SIGUSR2 时，最后一行输出为_____

2．当 N=3 时，A/B 为 SIGUSR1 或 SIGUSR2，E 为 exit(0)时，共有_____行输出

a） A 为 SIGUSR1，B 为 SIGUSR1 时，最后一行输出为_____

b） A 为 SIGUSR1，B 为 SIGUSR2 时，最后一行输出为_____

c） A 为 SIGUSR2，B 为 SIGUSR1 时，最后一行输出为_____

d） A 为 SIGUSR2，B 为 SIGUSR2 时，最后一行输出为_____

得分

第六题（12 分）

通常处理器中的 MMU 通过页表实现虚拟内存地址到物理内存地址的转换，而 TLB 被用于提升地址转换的效率。然而 TLB 必须和页表之间保持一致，才能保证 MMU 正确按页表转换虚拟地址。当修改页表内容时，需要通过 invlpg 指令将对应的 TLB 表项置为失效，以确保 TLB 与页表一致。

指令 invlpg m 把包含虚拟地址 m 的页面在 TLB 中的表项置为失效。

假设当前状态下，TLB 与页表是一致的。部分 TLB 表项如下（假设 TLB 是全相联的）：

有效位	TLB 标记	页面号
1	0x8040201	0x4801
1	0x8040233	0x4812
0	0x8040382	0x9C33
1	0x8046740	0x4801
0	0x8046621	0x8845
1	0x80467CD	0x6734

与上述 TLB 表项相关的两个页表页的地址及其中的部分页表项如下：

索引号	页面号	存在位
	Bits: 51 - 12	Bit: 0
0x1	0x4801	1
0x33	0x4812	1
0x54	0x8745	1
0x180	0x3212	1
0x182	0x9C33	1
0x1A1	0x9078	1

基地址为 0x4801000 的页表页

索引号	页面号	存在位
	Bits: 51 - 12	Bit: 0
0x10	0x2312	1
0x21	0x8845	1
0x73	0x4521	1
0x140	0x4801	1
0x182	0x8ACD	1
0x1CD	0x6734	1

基地址为 0x4812000 的页表页

设处理器采用的是 64 位虚拟地址和 64 位物理地址，页面大小为 4KB，页表为 4 级。

1．分析下面的指令序列：

```
1. movq $0x8040233000, %rbx
2. movq 0xA00(%rbx), %rcx
3. addq 0x11000, %rcx
4. movq %rcx, 0x108(rbx)
5. movq 0x8046621108, %rax
```

a) 执行完第 2 行指令后，载入到%rcx 中的物理页面号为:_____；

b) 执行完第 3 行指令后，%rcx 中的物理页面号为：_____；

c) 执行完第 5 行指令后，%rcx 和%rax 中的内容之间有何关系？_____

d) 在执行上述 5 行指令过程中，会发生_____次 TLB miss;

e) 在执行上述 5 行指令过程中，会发生_____次 page fault;

2．接着上面的的 5 行指令接着执行下面的指令:

```
6. movq 0x8040382C10, %r10
7. movq %rax, 0x8046740C10
8. movq 0x8040382C10, %r11
9. invlpg 0x8040382C10
10. movq 0x8040382C10, %r12
```

a) 全部 10 条指令执行后，%r10 和%r11 中的内容之间有何关系？_____

b) 全部 10 条指令执行后，%r11 和%r12 中的内容之间有何关系？_____

c) 在执行上述 10 行指令过程中，共发生了_____次 TLB miss.

3．请写出执行完上述 10 行指令后，TLB 和页表页中有变化的表项中的内容。

有效位	TLB 标记	页面号
	0x8040201	
	0x8040233	
	0x8040382	
	0x8046740	
	0x8046621	
	0x80467CD	

索引号	页面号	存在位
	Bits: 51 - 12	Bit: 0
0x1		
0x33		
0x54		
0x180		
0x182		
0x1A1		

基地址为 0x4801000 的页表页

索引号	页面号	存在位
	Bits: 51 - 12	Bit: 0
0x10		
0x21		
0x73		
0x140		
0x182		
0x1CD		

基地址为 0x4812000 的页表页