НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №7**

*з дисципліни* ***«****Архітектура комп’ютерів 2****»***

Виконав:

студент 3 курсу

групи ІВ-81

Соколов В.С.

Перевірив:

Каплунов А. В.

Київ 2020 р.

**Лістинг програми:**

**hello1.c**

#include <linux/init.h>

#include <linux/module.h>

#include <linux/printk.h>

#include <linux/types.h>

#include <linux/slab.h>

#include <linux/ktime.h>

#include <hello1.h>

MODULE\_LICENSE("Dual BSD/GPL");

MODULE\_AUTHOR("Volodymyr Sokolov");

MODULE\_DESCRIPTION("AK-2 lab\_7: hello1.c");

struct timeit\_list {

struct list\_head node;

ktime\_t before;

ktime\_t after;

};

static struct list\_head head\_node = LIST\_HEAD\_INIT(head\_node);

int print\_hello(uint n)

{

struct timeit\_list \*list, \*tmp;

uint i;

BUG\_ON(n > 10);

if (n <= 0) {

pr\_err("ERROR! n < 0\n");

return -EINVAL;

} else if (n == 0) {

pr\_warn("WARNING! n = 0\n");

} else if (n >= 5 && n <= 10) {

pr\_warn("WARNING! 5 <= n <= 10\n");

}

for (i = 0; i < n; i++) {

list = kmalloc(sizeof(struct timeit\_list), GFP\_KERNEL);

if (i == 7)

list = NULL;

if (ZERO\_OR\_NULL\_PTR(list))

goto clean\_up;

list->before = ktime\_get();

pr\_info("Hello, world!\n");

list->after = ktime\_get();

list\_add\_tail(&list->node, &head\_node);

}

return 0;

clean\_up:

list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {

list\_del(&list->node);

kfree(list);

}

pr\_err("ERROR! Memory is out\n");

return -ENOMEM;

}

EXPORT\_SYMBOL(print\_hello);

static int \_\_init init\_hello(void)

{

pr\_info("hello1 init\n");

return 0;

}

static void \_\_exit exit\_hello(void)

{

struct timeit\_list \*list, \*tmp;

list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {

pr\_info("Time: %lld", list->after - list->before);

list\_del(&list->node);

kfree(list);

}

BUG\_ON(!list\_empty(&head\_node));

pr\_info("hello1 exit\n");

}

module\_init(init\_hello);

module\_exit(exit\_hello);

**hello2.c**

#include <linux/init.h>

#include <linux/module.h>

#include <linux/printk.h>

#include <linux/types.h>

#include <linux/slab.h>

#include <linux/ktime.h>

#include <hello1.h>

MODULE\_LICENSE("Dual BSD/GPL");

MODULE\_DESCRIPTION("AK-2 lab\_7: hello2.c");

MODULE\_AUTHOR("Volodymyr Sokolov");

static uint n = 1;

module\_param(n, uint, 0);

MODULE\_PARM\_DESC(n, "How many hellos to print\n");

static int \_\_init init\_hello(void)

{

pr\_info("hello2 init\n");

print\_hello(n);

return 0;

}

static void \_\_exit exit\_hello(void)

{

pr\_info("hello2 exit\n");

}

module\_init(init\_hello);

module\_exit(exit\_hello);

**Makefile**

ccflags-y := -I$(PWD)/inc

ifneq ($(KERNELRELEASE),)

# kbuild part of makefile

obj-m := hello1.o hello2.o

ccflags-y += -g -DDEBUG

else

# normal makefile

KDIR ?= /lib/modules/`uname -r`/build

default:

$(MAKE) -C $(KDIR) M=$$PWD

cp hello1.ko hello1.ko.unstripped

cp hello2.ko hello2.ko.unstripped

$(CROSS\_COMPILE)strip -g hello1.ko

$(CROSS\_COMPILE)strip -g hello2.ko

clean:

$(MAKE) -C $(KDIR) M=$$PWD clean

%.s %.i: %.c

$(MAKE) -C $(KDIR) M=$$PWD $@

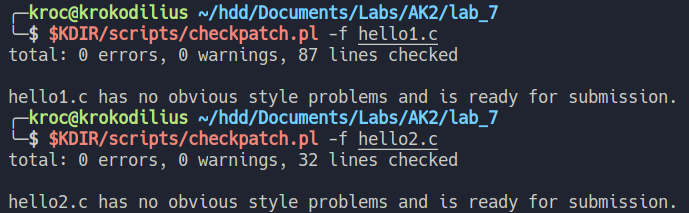
endif

**inc/hello1.h**

#include <linux/types.h>

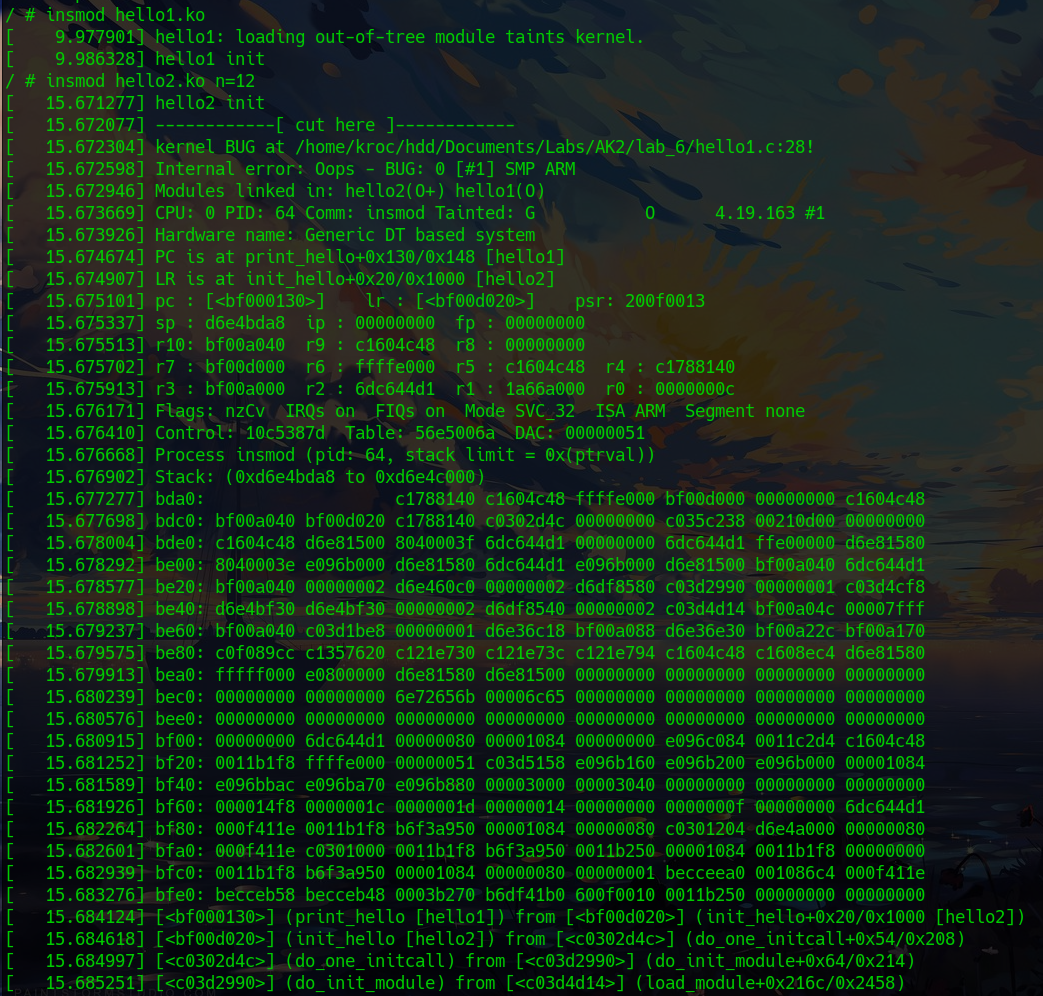
int print\_hello(uint n);

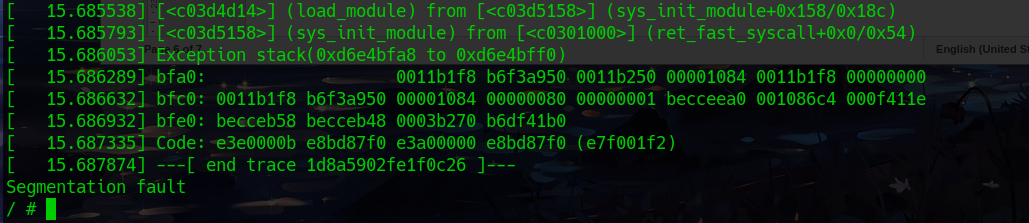
**Відповідність Linux Kernel Coding Style:**



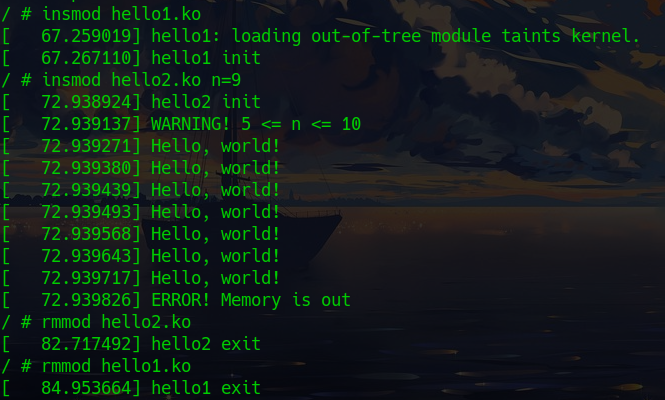
**Скріншоти виконання:**

1. *insmod hello2.ko n=12 (перевірка роботи BUG\_ON)*





*2. Виконання з параметром n=8 призводить до помилки (ставимо NULL)*



3. Дизассемблерний код

***hello1.ko***

hello1.ko.unstripped: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <print\_hello>:

static struct list\_head head\_node = LIST\_HEAD\_INIT(head\_node);

int print\_hello(uint n)

{

0: e8 00 00 00 00 callq 5 <print\_hello+0x5>

struct timeit\_list \*list, \*tmp;

uint i;

BUG\_ON(n > 10);

5: 83 ff 0a cmp $0xa,%edi

8: 0f 87 0b 01 00 00 ja 119 <print\_hello+0x119>

{

e: 55 push %rbp

f: 48 89 e5 mov %rsp,%rbp

12: 41 55 push %r13

14: 41 54 push %r12

16: 41 89 fc mov %edi,%r12d

19: 53 push %rbx

if (n <= 0) {

1a: 85 ff test %edi,%edi

1c: 0f 84 00 00 00 00 je 22 <print\_hello+0x22>

pr\_err("ERROR! n < 0\n");

return -EINVAL;

} else if (n == 0) {

pr\_warn("WARNING! n = 0\n");

} else if (n >= 5 && n <= 10) {

22: 8d 47 fb lea -0x5(%rdi),%eax

25: 83 f8 05 cmp $0x5,%eax

28: 0f 86 00 00 00 00 jbe 2e <print\_hello+0x2e>

index = kmalloc\_index(size);

if (!index)

return ZERO\_SIZE\_PTR;

return kmem\_cache\_alloc\_trace(

2e: ba 20 00 00 00 mov $0x20,%edx

33: 48 8b 3d 00 00 00 00 mov 0x0(%rip),%rdi # 3a <print\_hello+0x3a>

3a: be c0 0c 00 00 mov $0xcc0,%esi

3f: 41 bd 08 00 00 00 mov $0x8,%r13d

45: 45 29 e5 sub %r12d,%r13d

48: 41 bc 07 00 00 00 mov $0x7,%r12d

4e: e8 00 00 00 00 callq 53 <print\_hello+0x53>

53: 48 89 c3 mov %rax,%rbx

for (i = 0; i < n; i++) {

list = kmalloc(sizeof(struct timeit\_list), GFP\_KERNEL);

if (i == 7)

list = NULL;

if (ZERO\_OR\_NULL\_PTR(list))

56: 48 83 fb 10 cmp $0x10,%rbx

5a: 76 5e jbe ba <print\_hello+0xba>

goto clean\_up;

list->before = ktime\_get();

5c: e8 00 00 00 00 callq 61 <print\_hello+0x61>

pr\_info("Hello, world!\n");

61: 48 c7 c7 00 00 00 00 mov $0x0,%rdi

list->before = ktime\_get();

68: 48 89 43 10 mov %rax,0x10(%rbx)

pr\_info("Hello, world!\n");

6c: e8 00 00 00 00 callq 71 <print\_hello+0x71>

list->after = ktime\_get();

71: e8 00 00 00 00 callq 76 <print\_hello+0x76>

{

if (!\_\_list\_add\_valid(new, prev, next))

return;

next->prev = new;

new->next = next;

76: 48 c7 03 00 00 00 00 movq $0x0,(%rbx)

7d: 48 89 43 18 mov %rax,0x18(%rbx)

\* Insert a new entry before the specified head.

\* This is useful for implementing queues.

\*/

static inline void list\_add\_tail(struct list\_head \*new, struct list\_head \*head)

{

\_\_list\_add(new, head->prev, head);

81: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # 88 <print\_hello+0x88>

next->prev = new;

88: 48 89 1d 00 00 00 00 mov %rbx,0x0(%rip) # 8f <print\_hello+0x8f>

new->prev = prev;

8f: 48 89 43 08 mov %rax,0x8(%rbx)

{

switch (size) {

case 1: \*(volatile \_\_u8 \*)p = \*(\_\_u8 \*)res; break;

case 2: \*(volatile \_\_u16 \*)p = \*(\_\_u16 \*)res; break;

case 4: \*(volatile \_\_u32 \*)p = \*(\_\_u32 \*)res; break;

case 8: \*(volatile \_\_u64 \*)p = \*(\_\_u64 \*)res; break;

93: 48 89 18 mov %rbx,(%rax)

for (i = 0; i < n; i++) {

96: 45 39 ec cmp %r13d,%r12d

99: 74 75 je 110 <print\_hello+0x110>

9b: 48 8b 3d 00 00 00 00 mov 0x0(%rip),%rdi # a2 <print\_hello+0xa2>

a2: ba 20 00 00 00 mov $0x20,%edx

a7: be c0 0c 00 00 mov $0xcc0,%esi

ac: e8 00 00 00 00 callq b1 <print\_hello+0xb1>

b1: 48 89 c3 mov %rax,%rbx

if (i == 7)

b4: 41 83 ec 01 sub $0x1,%r12d

b8: 75 9c jne 56 <print\_hello+0x56>

list\_add\_tail(&list->node, &head\_node);

}

return 0;

clean\_up:

list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {

ba: 48 8b 3d 00 00 00 00 mov 0x0(%rip),%rdi # c1 <print\_hello+0xc1>

c1: 48 8b 1f mov (%rdi),%rbx

c4: 48 81 ff 00 00 00 00 cmp $0x0,%rdi

cb: 0f 84 00 00 00 00 je d1 <print\_hello+0xd1>

}

static inline void list\_del(struct list\_head \*entry)

{

\_\_list\_del\_entry(entry);

entry->next = LIST\_POISON1;

d1: 49 bd 00 01 00 00 00 movabs $0xdead000000000100,%r13

d8: 00 ad de

entry->prev = LIST\_POISON2;

db: 49 bc 22 01 00 00 00 movabs $0xdead000000000122,%r12

e2: 00 ad de

\_\_list\_del\_entry(entry);

e5: 48 8b 47 08 mov 0x8(%rdi),%rax

next->prev = prev;

e9: 48 89 43 08 mov %rax,0x8(%rbx)

ed: 48 89 18 mov %rbx,(%rax)

entry->next = LIST\_POISON1;

f0: 4c 89 2f mov %r13,(%rdi)

entry->prev = LIST\_POISON2;

f3: 4c 89 67 08 mov %r12,0x8(%rdi)

list\_del(&list->node);

kfree(list);

f7: e8 00 00 00 00 callq fc <print\_hello+0xfc>

list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {

fc: 48 89 df mov %rbx,%rdi

ff: 48 8b 1b mov (%rbx),%rbx

102: 48 81 ff 00 00 00 00 cmp $0x0,%rdi

109: 75 da jne e5 <print\_hello+0xe5>

10b: e9 00 00 00 00 jmpq 110 <print\_hello+0x110>

return 0;

110: 31 c0 xor %eax,%eax

}

pr\_err("ERROR! Memory is out\n");

return -ENOMEM;

}

112: 5b pop %rbx

113: 41 5c pop %r12

115: 41 5d pop %r13

117: 5d pop %rbp

118: c3 retq

BUG\_ON(n > 10);

119: 0f 0b ud2

Disassembly of section .text.unlikely:

0000000000000000 <print\_hello.cold>:

pr\_err("ERROR! Memory is out\n");

0: 48 c7 c7 00 00 00 00 mov $0x0,%rdi

7: e8 00 00 00 00 callq c <print\_hello.cold+0xc>

return -ENOMEM;

c: b8 f4 ff ff ff mov $0xfffffff4,%eax

11: e9 00 00 00 00 jmpq 16 <print\_hello.cold+0x16>

pr\_warn("WARNING! 5 <= n <= 10\n");

16: 48 c7 c7 00 00 00 00 mov $0x0,%rdi

1d: e8 00 00 00 00 callq 22 <print\_hello.cold+0x22>

22: e9 00 00 00 00 jmpq 27 <print\_hello.cold+0x27>

pr\_err("ERROR! n < 0\n");

27: 48 c7 c7 00 00 00 00 mov $0x0,%rdi

2e: e8 00 00 00 00 callq 33 <print\_hello.cold+0x33>

return -EINVAL;

33: b8 ea ff ff ff mov $0xffffffea,%eax

38: e9 00 00 00 00 jmpq 3d <\_\_UNIQUE\_ID\_author38+0xb>

Disassembly of section .init.text:

0000000000000000 <init\_module>:

EXPORT\_SYMBOL(print\_hello);

static int \_\_init init\_hello(void)

{

0: e8 00 00 00 00 callq 5 <init\_module+0x5>

5: 55 push %rbp

pr\_info("hello1 init\n");

6: 48 c7 c7 00 00 00 00 mov $0x0,%rdi

{

d: 48 89 e5 mov %rsp,%rbp

pr\_info("hello1 init\n");

10: e8 00 00 00 00 callq 15 <init\_module+0x15>

return 0;

}

15: 31 c0 xor %eax,%eax

17: 5d pop %rbp

18: c3 retq

Disassembly of section .exit.text:

0000000000000000 <cleanup\_module>:

static void \_\_exit exit\_hello(void)

{

0: 55 push %rbp

1: 48 89 e5 mov %rsp,%rbp

4: 41 56 push %r14

entry->next = LIST\_POISON1;

6: 49 be 00 01 00 00 00 movabs $0xdead000000000100,%r14

d: 00 ad de

10: 41 55 push %r13

entry->prev = LIST\_POISON2;

12: 49 bd 22 01 00 00 00 movabs $0xdead000000000122,%r13

19: 00 ad de

1c: 41 54 push %r12

struct timeit\_list \*list, \*tmp;

list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {

1e: 4c 8b 25 00 00 00 00 mov 0x0(%rip),%r12 # 25 <cleanup\_module+0x25>

{

25: 53 push %rbx

list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {

26: 49 8b 1c 24 mov (%r12),%rbx

2a: 49 81 fc 00 00 00 00 cmp $0x0,%r12

31: 74 3f je 72 <cleanup\_module+0x72>

pr\_info("Time: %lld", list->after - list->before);

33: 49 8b 74 24 18 mov 0x18(%r12),%rsi

38: 48 c7 c7 00 00 00 00 mov $0x0,%rdi

3f: 49 2b 74 24 10 sub 0x10(%r12),%rsi

44: e8 00 00 00 00 callq 49 <cleanup\_module+0x49>

\_\_list\_del\_entry(entry);

49: 49 8b 14 24 mov (%r12),%rdx

4d: 49 8b 44 24 08 mov 0x8(%r12),%rax

list\_del(&list->node);

kfree(list);

52: 4c 89 e7 mov %r12,%rdi

next->prev = prev;

55: 48 89 42 08 mov %rax,0x8(%rdx)

59: 48 89 10 mov %rdx,(%rax)

entry->next = LIST\_POISON1;

5c: 4d 89 34 24 mov %r14,(%r12)

entry->prev = LIST\_POISON2;

60: 4d 89 6c 24 08 mov %r13,0x8(%r12)

list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {

65: 49 89 dc mov %rbx,%r12

kfree(list);

68: e8 00 00 00 00 callq 6d <cleanup\_module+0x6d>

list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {

6d: 48 8b 1b mov (%rbx),%rbx

70: eb b8 jmp 2a <cleanup\_module+0x2a>

\_\_READ\_ONCE\_SIZE;

72: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # 79 <cleanup\_module+0x79>

}

BUG\_ON(!list\_empty(&head\_node));

79: 48 3d 00 00 00 00 cmp $0x0,%rax

7f: 74 02 je 83 <cleanup\_module+0x83>

81: 0f 0b ud2

pr\_info("hello1 exit\n");

83: 48 c7 c7 00 00 00 00 mov $0x0,%rdi

8a: e8 00 00 00 00 callq 8f <cleanup\_module+0x8f>

}

8f: 5b pop %rbx

90: 41 5c pop %r12

92: 41 5d pop %r13

94: 41 5e pop %r14

96: 5d pop %rbp

97: c3 retq

***hello2.ko***

*hello1.ko.unstripped: file format elf64-x86-64*

*Disassembly of section .text:*

*0000000000000000 <print\_hello>:*

*static struct list\_head head\_node = LIST\_HEAD\_INIT(head\_node);*

*int print\_hello(uint n)*

*{*

*0: e8 00 00 00 00 callq 5 <print\_hello+0x5>*

*struct timeit\_list \*list, \*tmp;*

*uint i;*

*BUG\_ON(n > 10);*

*5: 83 ff 0a cmp $0xa,%edi*

*8: 0f 87 0b 01 00 00 ja 119 <print\_hello+0x119>*

*{*

*e: 55 push %rbp*

*f: 48 89 e5 mov %rsp,%rbp*

*12: 41 55 push %r13*

*14: 41 54 push %r12*

*16: 41 89 fc mov %edi,%r12d*

*19: 53 push %rbx*

*if (n <= 0) {*

*1a: 85 ff test %edi,%edi*

*1c: 0f 84 00 00 00 00 je 22 <print\_hello+0x22>*

*pr\_err("ERROR! n < 0\n");*

*return -EINVAL;*

*} else if (n == 0) {*

*pr\_warn("WARNING! n = 0\n");*

*} else if (n >= 5 && n <= 10) {*

*22: 8d 47 fb lea -0x5(%rdi),%eax*

*25: 83 f8 05 cmp $0x5,%eax*

*28: 0f 86 00 00 00 00 jbe 2e <print\_hello+0x2e>*

*index = kmalloc\_index(size);*

*if (!index)*

*return ZERO\_SIZE\_PTR;*

*return kmem\_cache\_alloc\_trace(*

*2e: ba 20 00 00 00 mov $0x20,%edx*

*33: 48 8b 3d 00 00 00 00 mov 0x0(%rip),%rdi # 3a <print\_hello+0x3a>*

*3a: be c0 0c 00 00 mov $0xcc0,%esi*

*3f: 41 bd 08 00 00 00 mov $0x8,%r13d*

*45: 45 29 e5 sub %r12d,%r13d*

*48: 41 bc 07 00 00 00 mov $0x7,%r12d*

*4e: e8 00 00 00 00 callq 53 <print\_hello+0x53>*

*53: 48 89 c3 mov %rax,%rbx*

*for (i = 0; i < n; i++) {*

*list = kmalloc(sizeof(struct timeit\_list), GFP\_KERNEL);*

*if (i == 7)*

*list = NULL;*

*if (ZERO\_OR\_NULL\_PTR(list))*

*56: 48 83 fb 10 cmp $0x10,%rbx*

*5a: 76 5e jbe ba <print\_hello+0xba>*

*goto clean\_up;*

*list->before = ktime\_get();*

*5c: e8 00 00 00 00 callq 61 <print\_hello+0x61>*

*pr\_info("Hello, world!\n");*

*61: 48 c7 c7 00 00 00 00 mov $0x0,%rdi*

*list->before = ktime\_get();*

*68: 48 89 43 10 mov %rax,0x10(%rbx)*

*pr\_info("Hello, world!\n");*

*6c: e8 00 00 00 00 callq 71 <print\_hello+0x71>*

*list->after = ktime\_get();*

*71: e8 00 00 00 00 callq 76 <print\_hello+0x76>*

*{*

*if (!\_\_list\_add\_valid(new, prev, next))*

*return;*

*next->prev = new;*

*new->next = next;*

*76: 48 c7 03 00 00 00 00 movq $0x0,(%rbx)*

*7d: 48 89 43 18 mov %rax,0x18(%rbx)*

*\* Insert a new entry before the specified head.*

*\* This is useful for implementing queues.*

*\*/*

*static inline void list\_add\_tail(struct list\_head \*new, struct list\_head \*head)*

*{*

*\_\_list\_add(new, head->prev, head);*

*81: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # 88 <print\_hello+0x88>*

*next->prev = new;*

*88: 48 89 1d 00 00 00 00 mov %rbx,0x0(%rip) # 8f <print\_hello+0x8f>*

*new->prev = prev;*

*8f: 48 89 43 08 mov %rax,0x8(%rbx)*

*{*

*switch (size) {*

*case 1: \*(volatile \_\_u8 \*)p = \*(\_\_u8 \*)res; break;*

*case 2: \*(volatile \_\_u16 \*)p = \*(\_\_u16 \*)res; break;*

*case 4: \*(volatile \_\_u32 \*)p = \*(\_\_u32 \*)res; break;*

*case 8: \*(volatile \_\_u64 \*)p = \*(\_\_u64 \*)res; break;*

*93: 48 89 18 mov %rbx,(%rax)*

*for (i = 0; i < n; i++) {*

*96: 45 39 ec cmp %r13d,%r12d*

*99: 74 75 je 110 <print\_hello+0x110>*

*9b: 48 8b 3d 00 00 00 00 mov 0x0(%rip),%rdi # a2 <print\_hello+0xa2>*

*a2: ba 20 00 00 00 mov $0x20,%edx*

*a7: be c0 0c 00 00 mov $0xcc0,%esi*

*ac: e8 00 00 00 00 callq b1 <print\_hello+0xb1>*

*b1: 48 89 c3 mov %rax,%rbx*

*if (i == 7)*

*b4: 41 83 ec 01 sub $0x1,%r12d*

*b8: 75 9c jne 56 <print\_hello+0x56>*

*list\_add\_tail(&list->node, &head\_node);*

*}*

*return 0;*

*clean\_up:*

*list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {*

*ba: 48 8b 3d 00 00 00 00 mov 0x0(%rip),%rdi # c1 <print\_hello+0xc1>*

*c1: 48 8b 1f mov (%rdi),%rbx*

*c4: 48 81 ff 00 00 00 00 cmp $0x0,%rdi*

*cb: 0f 84 00 00 00 00 je d1 <print\_hello+0xd1>*

*}*

*static inline void list\_del(struct list\_head \*entry)*

*{*

*\_\_list\_del\_entry(entry);*

*entry->next = LIST\_POISON1;*

*d1: 49 bd 00 01 00 00 00 movabs $0xdead000000000100,%r13*

*d8: 00 ad de*

*entry->prev = LIST\_POISON2;*

*db: 49 bc 22 01 00 00 00 movabs $0xdead000000000122,%r12*

*e2: 00 ad de*

*\_\_list\_del\_entry(entry);*

*e5: 48 8b 47 08 mov 0x8(%rdi),%rax*

*next->prev = prev;*

*e9: 48 89 43 08 mov %rax,0x8(%rbx)*

*ed: 48 89 18 mov %rbx,(%rax)*

*entry->next = LIST\_POISON1;*

*f0: 4c 89 2f mov %r13,(%rdi)*

*entry->prev = LIST\_POISON2;*

*f3: 4c 89 67 08 mov %r12,0x8(%rdi)*

*list\_del(&list->node);*

*kfree(list);*

*f7: e8 00 00 00 00 callq fc <print\_hello+0xfc>*

*list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {*

*fc: 48 89 df mov %rbx,%rdi*

*ff: 48 8b 1b mov (%rbx),%rbx*

*102: 48 81 ff 00 00 00 00 cmp $0x0,%rdi*

*109: 75 da jne e5 <print\_hello+0xe5>*

*10b: e9 00 00 00 00 jmpq 110 <print\_hello+0x110>*

*return 0;*

*110: 31 c0 xor %eax,%eax*

*}*

*pr\_err("ERROR! Memory is out\n");*

*return -ENOMEM;*

*}*

*112: 5b pop %rbx*

*113: 41 5c pop %r12*

*115: 41 5d pop %r13*

*117: 5d pop %rbp*

*118: c3 retq*

*BUG\_ON(n > 10);*

*119: 0f 0b ud2*

*Disassembly of section .text.unlikely:*

*0000000000000000 <print\_hello.cold>:*

*pr\_err("ERROR! Memory is out\n");*

*0: 48 c7 c7 00 00 00 00 mov $0x0,%rdi*

*7: e8 00 00 00 00 callq c <print\_hello.cold+0xc>*

*return -ENOMEM;*

*c: b8 f4 ff ff ff mov $0xfffffff4,%eax*

*11: e9 00 00 00 00 jmpq 16 <print\_hello.cold+0x16>*

*pr\_warn("WARNING! 5 <= n <= 10\n");*

*16: 48 c7 c7 00 00 00 00 mov $0x0,%rdi*

*1d: e8 00 00 00 00 callq 22 <print\_hello.cold+0x22>*

*22: e9 00 00 00 00 jmpq 27 <print\_hello.cold+0x27>*

*pr\_err("ERROR! n < 0\n");*

*27: 48 c7 c7 00 00 00 00 mov $0x0,%rdi*

*2e: e8 00 00 00 00 callq 33 <print\_hello.cold+0x33>*

*return -EINVAL;*

*33: b8 ea ff ff ff mov $0xffffffea,%eax*

*38: e9 00 00 00 00 jmpq 3d <\_\_UNIQUE\_ID\_author38+0xb>*

*Disassembly of section .init.text:*

*0000000000000000 <init\_module>:*

*EXPORT\_SYMBOL(print\_hello);*

*static int \_\_init init\_hello(void)*

*{*

*0: e8 00 00 00 00 callq 5 <init\_module+0x5>*

*5: 55 push %rbp*

*pr\_info("hello1 init\n");*

*6: 48 c7 c7 00 00 00 00 mov $0x0,%rdi*

*{*

*d: 48 89 e5 mov %rsp,%rbp*

*pr\_info("hello1 init\n");*

*10: e8 00 00 00 00 callq 15 <init\_module+0x15>*

*return 0;*

*}*

*15: 31 c0 xor %eax,%eax*

*17: 5d pop %rbp*

*18: c3 retq*

*Disassembly of section .exit.text:*

*0000000000000000 <cleanup\_module>:*

*static void \_\_exit exit\_hello(void)*

*{*

*0: 55 push %rbp*

*1: 48 89 e5 mov %rsp,%rbp*

*4: 41 56 push %r14*

*entry->next = LIST\_POISON1;*

*6: 49 be 00 01 00 00 00 movabs $0xdead000000000100,%r14*

*d: 00 ad de*

*10: 41 55 push %r13*

*entry->prev = LIST\_POISON2;*

*12: 49 bd 22 01 00 00 00 movabs $0xdead000000000122,%r13*

*19: 00 ad de*

*1c: 41 54 push %r12*

*struct timeit\_list \*list, \*tmp;*

*list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {*

*1e: 4c 8b 25 00 00 00 00 mov 0x0(%rip),%r12 # 25 <cleanup\_module+0x25>*

*{*

*25: 53 push %rbx*

*list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {*

*26: 49 8b 1c 24 mov (%r12),%rbx*

*2a: 49 81 fc 00 00 00 00 cmp $0x0,%r12*

*31: 74 3f je 72 <cleanup\_module+0x72>*

*pr\_info("Time: %lld", list->after - list->before);*

*33: 49 8b 74 24 18 mov 0x18(%r12),%rsi*

*38: 48 c7 c7 00 00 00 00 mov $0x0,%rdi*

*3f: 49 2b 74 24 10 sub 0x10(%r12),%rsi*

*44: e8 00 00 00 00 callq 49 <cleanup\_module+0x49>*

*\_\_list\_del\_entry(entry);*

*49: 49 8b 14 24 mov (%r12),%rdx*

*4d: 49 8b 44 24 08 mov 0x8(%r12),%rax*

*list\_del(&list->node);*

*kfree(list);*

*52: 4c 89 e7 mov %r12,%rdi*

*next->prev = prev;*

*55: 48 89 42 08 mov %rax,0x8(%rdx)*

*59: 48 89 10 mov %rdx,(%rax)*

*entry->next = LIST\_POISON1;*

*5c: 4d 89 34 24 mov %r14,(%r12)*

*entry->prev = LIST\_POISON2;*

*60: 4d 89 6c 24 08 mov %r13,0x8(%r12)*

*list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {*

*65: 49 89 dc mov %rbx,%r12*

*kfree(list);*

*68: e8 00 00 00 00 callq 6d <cleanup\_module+0x6d>*

*list\_for\_each\_entry\_safe(list, tmp, &head\_node, node) {*

*6d: 48 8b 1b mov (%rbx),%rbx*

*70: eb b8 jmp 2a <cleanup\_module+0x2a>*

*\_\_READ\_ONCE\_SIZE;*

*72: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # 79 <cleanup\_module+0x79>*

*}*

*BUG\_ON(!list\_empty(&head\_node));*

*79: 48 3d 00 00 00 00 cmp $0x0,%rax*

*7f: 74 02 je 83 <cleanup\_module+0x83>*

*81: 0f 0b ud2*

*pr\_info("hello1 exit\n");*

*83: 48 c7 c7 00 00 00 00 mov $0x0,%rdi*

*8a: e8 00 00 00 00 callq 8f <cleanup\_module+0x8f>*

*}*

*8f: 5b pop %rbx*

*90: 41 5c pop %r12*

*92: 41 5d pop %r13*

*94: 41 5e pop %r14*

*96: 5d pop %rbp*

*97: c3 retq*