

Lab 4: Integer Stack Kernel Module Implementation

Amir Gubaidullin CBS-01

am.gubaidullin@innopolis.university

May 4, 2025

Github repo: <https://github.com/dExNight/Advanced-Linux-Lab4>

1. Introduction

Linux kernel module that provides an integer stack data structure accessed through character device. The module supports basic stack operations (push/pop) and stack size configuration via ioctl system calls. User-space utility provides command-line interface for interacting with the kernel module

2. Objectives

- Implement a kernel module with dynamic memory allocation
- Implement thread-safe operations using mutex synchronization
- Create a character device driver with file operations
- Develop a user-space CLI utility for device interaction
- Handle edge cases and error conditions properly

3. Implementation

3.1 I created project structure

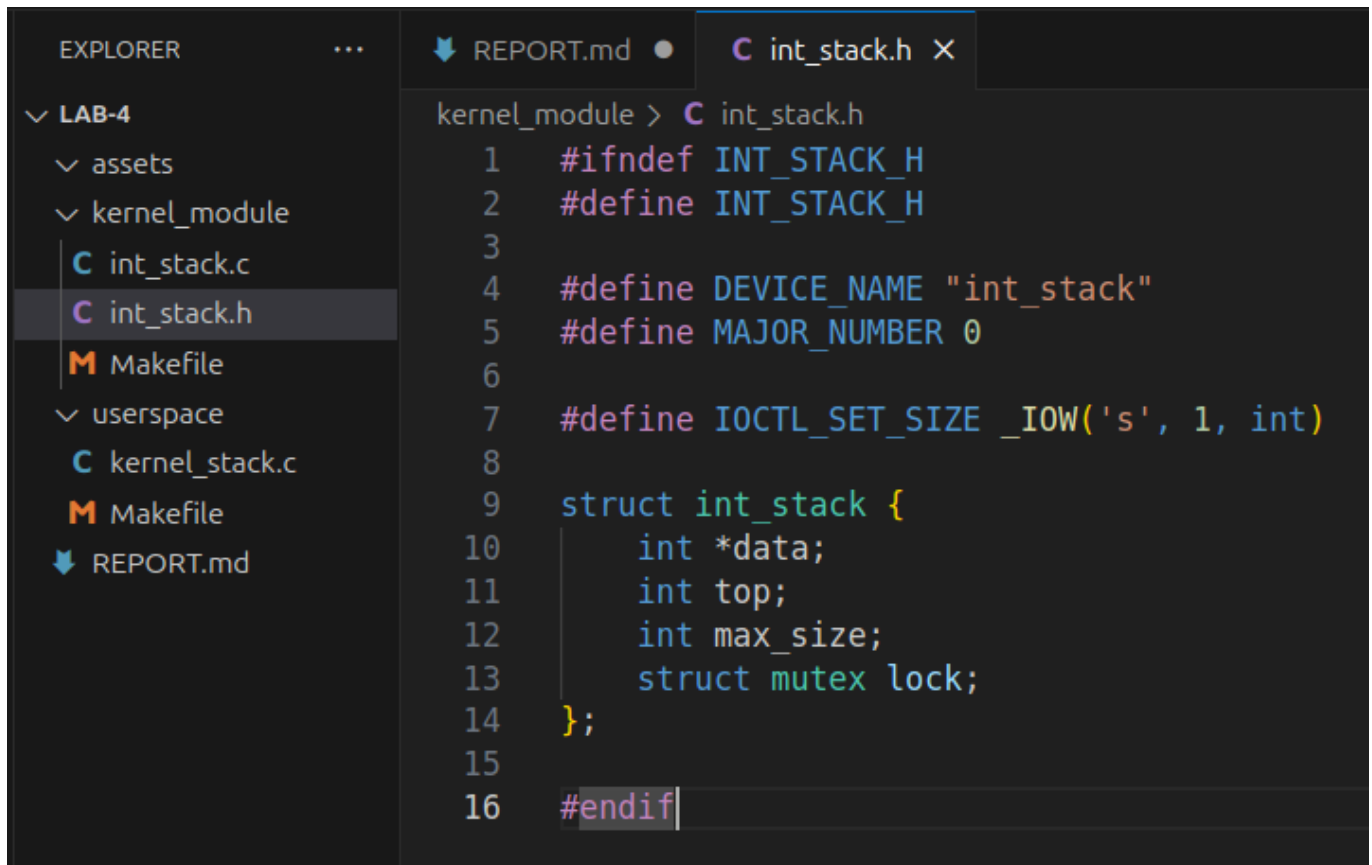
```
mkdir -p kernel_module userspace
```

The project is organized into two main components:

```
lab-4/  
├── kernel_module/  
│   ├── int_stack.h  
│   ├── int_stack.c  
│   └── Makefile  
└── userspace/  
    ├── kernel_stack.c  
    └── Makefile
```

3.2 Header file implementation

Hheader file (int_stack.h) defines the core data structures, constants:



The screenshot shows a code editor with a dark theme. On the left is the 'EXPLORER' sidebar showing a project structure for 'LAB-4'. The 'kernel_module' directory is expanded, showing 'int_stack.c', 'int_stack.h' (selected), and 'Makefile'. The main editor area shows the content of 'int_stack.h' with the following code:

```
kernel_module > C int_stack.h
1  #ifndef INT_STACK_H
2  #define INT_STACK_H
3
4  #define DEVICE_NAME "int_stack"
5  #define MAJOR_NUMBER 0
6
7  #define IOCTL_SET_SIZE _IOW('s', 1, int)
8
9  struct int_stack {
10     int *data;
11     int top;
12     int max_size;
13     struct mutex lock;
14 };
15
16 #endif
```

Key components:

- Device name definition for registration
- IOCTL command for stack size configuration
- Stack structure with dynamic array, top pointer, maximum size
- Mutex for thread synchronization

3.3 Kernel module implementation

The kernel module (int_stack.c) implements the character device driver:

```

C int_stack.c X
kernel_module > C int_stack.c
1  #include <linux/module.h>
2  #include <linux/kernel.h>
3  #include <linux/init.h>
4  #include <linux/fs.h>
5  #include <linux/device.h>
6  #include <linux/cdev.h>
7  #include <linux/slab.h>
8  #include <linux/uaccess.h>
9  #include <linux/mutex.h>
10
11 #include "int_stack.h"
12
13 MODULE_LICENSE("GPL");
14 MODULE_AUTHOR("Amir Gubaidullin");
15 MODULE_DESCRIPTION("Integer Stack Character Device");
16 MODULE_VERSION("0.1");
17
18 static dev_t dev_num;
19 static struct cdev c_dev;
20 static struct class *cl;
21 static struct int_stack *stack;
22
23 // Function prototypes
24 static int device_open(struct inode *, struct file *);
25 static int device_release(struct inode *, struct file *);
26 static ssize_t device_read(struct file *, char __user *, size_t, loff_t *);
27 static ssize_t device_write(struct file *, const char __user *, size_t, loff_t *);
28 static long device_ioctl(struct file *, unsigned int, unsigned long);
29
30 static struct file_operations fops = {
31     .open = device_open,
32     .release = device_release,
33     .read = device_read,
34     .write = device_write,
35     .unlocked_ioctl = device_ioctl,
36     .owner = THIS_MODULE

```

1. Module Initialization (`int_stack_init`):

- Dynamic device number allocation
- Device class creation for automatic device file generation
- Character device registration
- Stack structure allocation and initialization

2. File Operations:

- `open()` and `release()`: Basic file operations (currently minimal implementation)
- `read()`: Implements pop operation, returns 0 bytes when stack is empty
- `write()`: Implements push operation, returns -ERANGE when stack is full
- `ioctl()`: Configures stack size, validates input, manages memory allocation

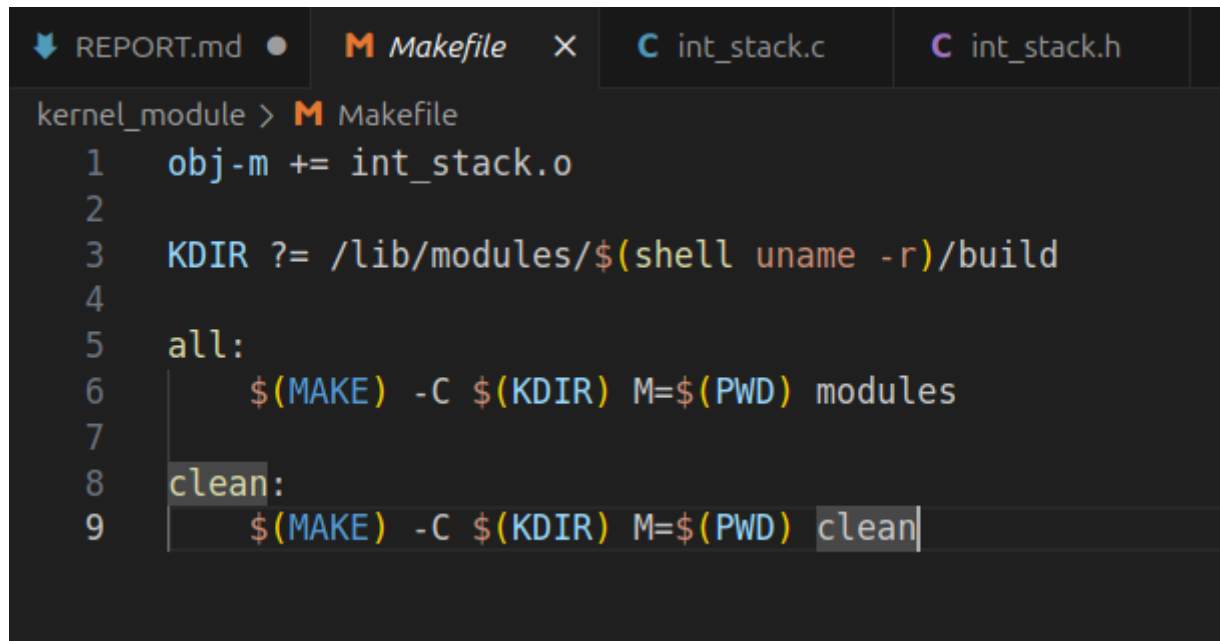
3. Thread Safety:

- All stack operations are protected by mutex locks
- Proper lock/unlock sequences in error paths
- Prevents race conditions during concurrent access

4. Module Cleanup (`int_stack_exit`):

- Frees allocated memory
- Unregisters character device
- Destroys device class
- Releases device number

Then created kernel module makefile:



The screenshot shows a code editor with four tabs: 'REPORT.md', 'Makefile', 'int_stack.c', and 'int_stack.h'. The 'Makefile' tab is active, displaying the following content:

```
kernel_module > M Makefile
1  obj-m += int_stack.o
2
3  KDIR ?= /lib/modules/$(shell uname -r)/build
4
5  all:
6      $(MAKE) -C $(KDIR) M=$(PWD) modules
7
8  clean:
9      $(MAKE) -C $(KDIR) M=$(PWD) clean
```

Features:

- Out-of-tree module compilation
- Automatic kernel version detection
- Clean target for build artifacts

3.4 User-space utility

The user-space program (kernel_stack.c) provides a CLI interface:

```

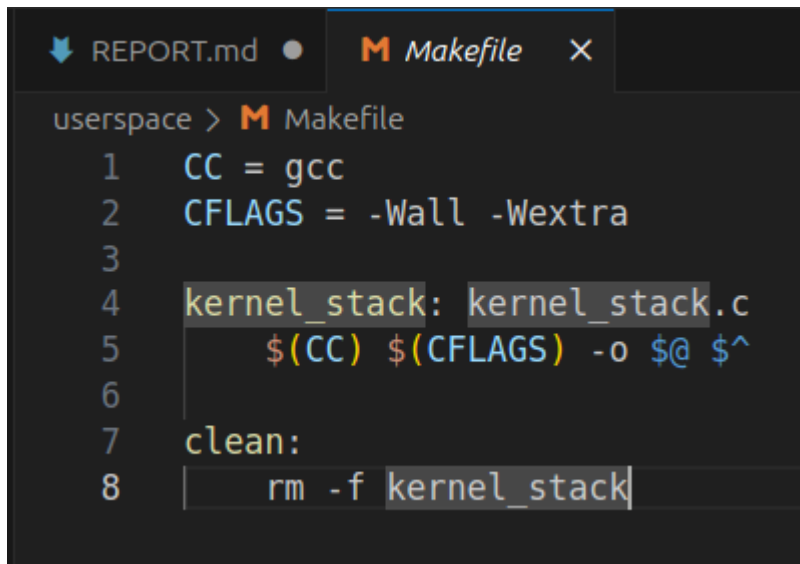
userspace > C kernel_stack.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <fcntl.h>
4  #include <unistd.h>
5  #include <sys/ioctl.h>
6  #include <string.h>
7  #include <errno.h>
8
9  #define DEVICE_PATH "/dev/int_stack"
10 #define IOCTL_SET_SIZE _IOW('s', 1, int)
11
12 void print_usage(const char *prog_name) {
13     printf("Usage:\n");
14     printf(" %s set-size <size>\n", prog_name);
15     printf(" %s push <value>\n", prog_name);
16     printf(" %s pop\n", prog_name);
17     printf(" %s unwind\n", prog_name);
18 }
19
20 int main(int argc, char *argv[]) {
21     int fd;
22     int value, size;
23     int ret;
24
25     if (argc < 2) {
26         print_usage(argv[0]);
27         return 1;
28     }
29
30     fd = open(DEVICE_PATH, O_RDWR);
31     if (fd < 0) {
32         perror("Failed to open device");
33         return errno;
34     }
35
36     if (strcmp(argv[1], "set-size") == 0) {
37         if (argc < 3) {

```

Supported commands:

- **set-size <size>**: Configure stack size
- **push <value>**: Add element to stack
- **pop**: Remove and display top element
- **unwind**: Remove and display all elements

Then created user-space makefile:



```

userspace > M Makefile
1  CC = gcc
2  CFLAGS = -Wall -Wextra
3
4  kernel_stack: kernel_stack.c
5      $(CC) $(CFLAGS) -o $@ $^
6
7  clean:
8      rm -f kernel_stack

```

Features:

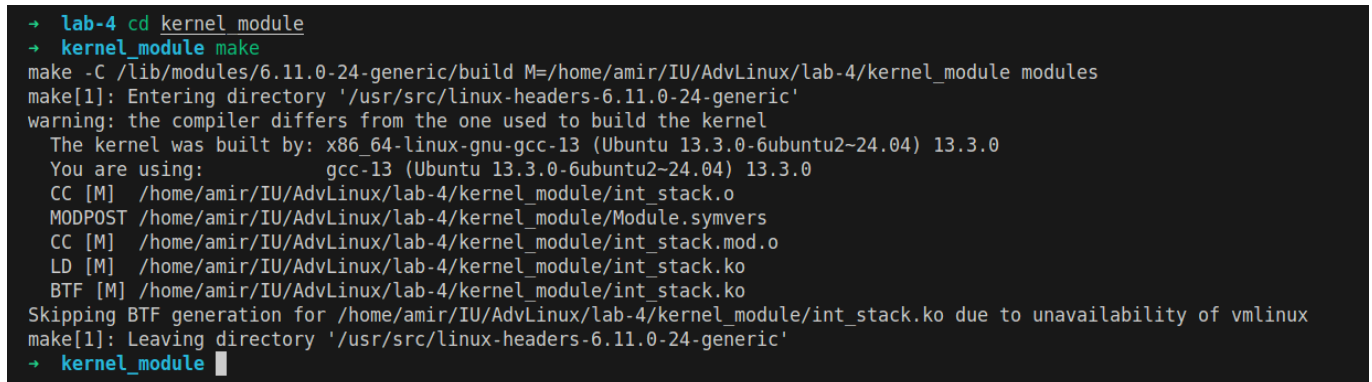
- Strict compilation flags (-Wall -Wextra)
- Simple build process for CLI utility

3.5 Building and testing

Successfully built both components:

- Kernel module compiled without warnings
- User-space utility compiled with strict flags

Built kernel module:

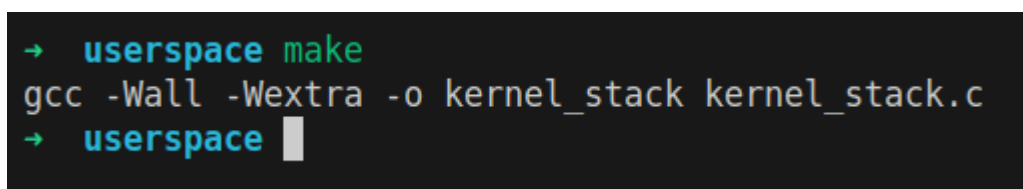


```

→ lab-4 cd kernel_module
→ kernel_module make
make -C /lib/modules/6.11.0-24-generic/build M=/home/amir/IU/AdvLinux/lab-4/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-6.11.0-24-generic'
warning: the compiler differs from the one used to build the kernel
The kernel was built by: x86_64-linux-gnu-gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
You are using:          gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
CC [M] /home/amir/IU/AdvLinux/lab-4/kernel_module/int_stack.o
MODPOST /home/amir/IU/AdvLinux/lab-4/kernel_module/Module.symvers
CC [M] /home/amir/IU/AdvLinux/lab-4/kernel_module/int_stack.mod.o
LD [M] /home/amir/IU/AdvLinux/lab-4/kernel_module/int_stack.ko
BTF [M] /home/amir/IU/AdvLinux/lab-4/kernel_module/int_stack.ko
Skipping BTF generation for /home/amir/IU/AdvLinux/lab-4/kernel_module/int_stack.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.11.0-24-generic'
→ kernel_module

```

Built user-space utility:



```

→ userspace make
gcc -Wall -Wextra -o kernel_stack kernel_stack.c
→ userspace

```

I faced some issues due to Secure Boot being enabled, the module required signing before loading:

```

# Create signing keys
cd ~

```

```
mkdir kernel-signing
cd kernel-signing
openssl req -new -x509 -newkey rsa:2048 -keyout MOK.priv -outform DER -out
MOK.der -nodes -days 36500 -subj "/CN=Local Kernel Module Signing/"
openssl x509 -in MOK.der -inform DER -outform PEM -out MOK.pem
sudo mokutil --import MOK.der

# Reboot and enroll MOK

# Sign the module
cd /home/amir/IU/AdvLinux/lab-4/kernel_module
sudo /usr/src/linux-headers-$(uname -r)/scripts/sign-file sha256 \
~/kernel-signing/MOK.priv \
~/kernel-signing/MOK.der \
int_stack.ko
```

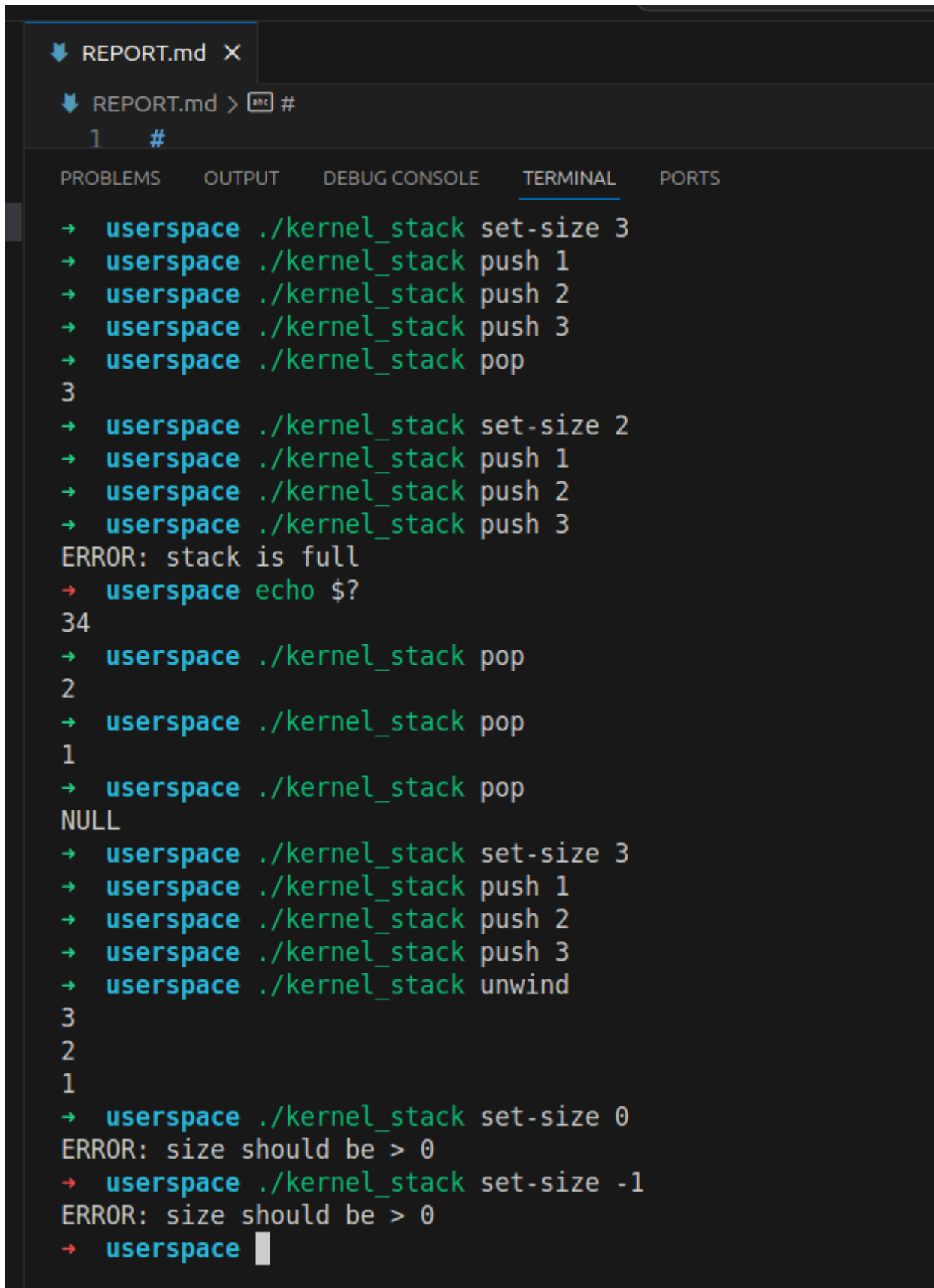
3.6 Module loading

After signing, the module was successfully loaded:

```
→ lab-4 cd /home/amir/IU/AdvLinux/lab-4/kernel_module
→ kernel_module sudo /usr/src/linux-headers-$(uname -r)/scripts/sign-file sha256 \
~/kernel-signing/MOK.priv \
~/kernel-signing/MOK.der \
int_stack.ko
→ kernel_module sudo insmod int_stack.ko
→ kernel_module sudo chmod 666 /dev/int_stack
→ kernel_module
```

3.7 Test cases

I ran test cases:



```
REPORT.md X
REPORT.md > [abc] #
1 #

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

→ userspace ./kernel_stack set-size 3
→ userspace ./kernel_stack push 1
→ userspace ./kernel_stack push 2
→ userspace ./kernel_stack push 3
→ userspace ./kernel_stack pop
3
→ userspace ./kernel_stack set-size 2
→ userspace ./kernel_stack push 1
→ userspace ./kernel_stack push 2
→ userspace ./kernel_stack push 3
ERROR: stack is full
→ userspace echo $?
34
→ userspace ./kernel_stack pop
2
→ userspace ./kernel_stack pop
1
→ userspace ./kernel_stack pop
NULL
→ userspace ./kernel_stack set-size 3
→ userspace ./kernel_stack push 1
→ userspace ./kernel_stack push 2
→ userspace ./kernel_stack push 3
→ userspace ./kernel_stack unwind
3
2
1
→ userspace ./kernel_stack set-size 0
ERROR: size should be > 0
→ userspace ./kernel_stack set-size -1
ERROR: size should be > 0
→ userspace
```

Executed test cases demonstrating:

1. Basic Operations:

- Setting stack size
- Push operations
- Pop operations

- Stack full error handling
- Stack empty handling

2. Edge Cases:

- Invalid size values (≤ 0)
- Stack overflow behavior
- Empty stack pop
- Unwind operation
- Error code verification

3.8 Resubmission

Check items are saved when the stack size increases

```

● → kernel_module sudo /usr/src/linux-headers-$(uname -r)/scripts/sign-file sha256 \
  ~/kernel-signing/MOK.priv \
  ~/kernel-signing/MOK.der \
  int_stack.ko
● → kernel_module sudo insmod int_stack.ko
● → kernel_module lsmod | grep int_stack
  int_stack          12288  0
● → kernel_module sudo chmod 666 /dev/int_stack
● → kernel_module cd ../userspace
● → userspace ./kernel_stack set-size 3
● → userspace ./kernel_stack push 1
● → userspace ./kernel_stack push 2
● → userspace ./kernel_stack push 3
● → userspace ./kernel_stack set-size 5
● → userspace ./kernel_stack unwind
  3
  2
  1
○ → userspace █

```

Check for deleting items when reducing the stack size

```
● → userspace ./kernel_stack set-size 5
● → userspace ./kernel_stack push 1
● → userspace ./kernel_stack push 2
● → userspace ./kernel_stack push 3
● → userspace ./kernel_stack push 4
● → userspace ./kernel_stack push 5
● → userspace ./kernel_stack set-size 3
● → userspace ./kernel_stack unwind
  3
  2
  1
○ → userspace █
```

Check correctness of operations after resizing

```
● → userspace ./kernel_stack set-size 2
● → userspace ./kernel_stack push 10
● → userspace ./kernel_stack push 20
⊕ → userspace ./kernel_stack push 30
  ERROR: stack is full
● → userspace ./kernel_stack pop
  20
● → userspace ./kernel_stack pop
  10
● → userspace ./kernel_stack pop
  NULL
○ → userspace █
```