ITF22519: Introduction to Operating Systems

Lab4: GNU Compiler, Debugging Tools, and SEGFAULT

Exercise 1

```
CC=gcc
CFLAGS=-I.

lab4: lab4.o message.o hello.o
	$(CC) -o lab4 lab4.o message.o hello.o
```

Makefile for linking lab4.c, message.c and hello.c and put it in an executable lab4 file.

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ make
gcc -I.   -c -o lab4.o lab4.c
gcc -I.   -c -o message.o message.c
gcc -I.   -c -o hello.o hello.c
gcc -o lab4 lab4.o message.o hello.o
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ nano message.c
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ nano hello.c
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
This is a lab section.
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
Goodbye
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
Goodbye
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
This is a lab section.
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
This is a lab section.
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
Goodbye
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
This is a lab section.
```

In this screenshot I have followed the instructions in the exercise text.

1. The “make” command compiles all files which have changed since last compilation. Since this is the first “make”, all files are compiled. Now the files “lab4.c”, “message.c” and “hello.c” have been linked into an executable file “lab4”.

2. We modify the files “message.c” and “hello.c”

3. Now we run the executable file “lab4” multiple times. Only the function “print_message()” is called, which is defined in “message.c”. Currently the “hello_world()” function is not called in “lab4.c”.

The changes we made in “message.c” and “hello.c” are also currently not included in our executable file “lab4”. This is because we have not executed the “make” command and recompiled our files.

In the next screenshot I have modified my approach, which produces a different result.

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ nano lab4.c
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ make
gcc -I.   -c -o lab4.o lab4.c
gcc -I.   -c -o message.o message.c
gcc -I.   -c -o hello.o hello.c
gcc -o lab4 lab4.o message.o hello.o
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
Hello mate!
Hello Lab4! :)
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
Hello mate!
Hello Lab4! :)
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
Hello mate!
Hello Lab4! :)
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
Goodbye friend :)
Hello Lab4! :)
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
This is a string of text
Hello Lab4! :)
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
This is a string of text
Hello Lab4! :)
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./lab4
Linking multiple c files...
See you later!
Hello Lab4! :)
```

1. I made some changes to the “lab4.c” file. Now we also call the “hello_world()” function, which is defined in “hello.c”.

2. We recompile the files and produce a new executable file “lab4” using the “make” command.

3. Now we can see that by executing our “lab4” file, we call both functions “print_message()” and “hello_world()”. We can also see that the changes we made in “message.c” and “hello.c” are included in our new executable.

Exercise 2

```
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ cc rand_string.c -o rand_string
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$ ./rand_string
Enter a line: hei
You entered the 3 character line: hei
The randomized string is
eieeiiieeh
*** stack smashing detected ***: terminated
Aborted (core dumped)
jrlundqv@IdeaPad5:~/OneDrive/22høst/OS/labs/lab4$
```

Compiling and running the program. Here we get a “stack smashing” error. This is a type of segmentation fault. The program is aborted because we are trying to access memory that does not “belong to us”.

```
int rand_array[10];
int length = 10;
for(i = 0; i <= length; i++)
{
        rand_array[i] = rand() % (n-1);
}
```

This is the part of the code that is causing the segfault. First, an array of size 10 is initialized. This means that the indexes of the array range from 0 to 9. Then an integer with value 10 is initialized.

Since the conditions for terminating the for loop is defined as “i <= length”, this means that the loop will terminate when i = 11, meaning it will complete its last iteration with the value i = 10.

What then happens is that at the last iteration, we try to access index 10 of the array. Since this isn’t a valid part of the array we initialized, we are trying to write something to a part of the memory which isn’t ours, hence the segmentation fault.

The fix is to remove the "=" operator from the for loop termination parameter, such that it reads "for(i = 0; i < length; i++)". Now the loop will terminate after the last index of the array has been written to, and we will not get a segmentation fault.

The output string always has 10 characters, independent of the input string. The reason is that the for loop that controls the output stream is terminated at i < length.

```
for(i = 0; i < length; i++)
{
        putchar(string[rand_array[i]]);
}
```