# CS:3620 — Spring 2020 – Homework 3

This homework is about the memory API in Linux.

*Please, ask general questions about the homework on ICON, so that everyone can benefit from the answers.*

**General Requirements** Submit your homework as a *single* `tar` file. When unpacked, the `tar` file must have the following directory structure (substitute ⟨your_HawkID⟩ with your actual HawkID):

```
⟨your_HawkID⟩
    task1
        compile.sh
        task1.c
    task2
        compile.sh
        task2.c
    task3
        compile.sh
        task3.c
```

The `.sh` files must have the `executable bit` set. No other file should be present in the archive.

You must write the source code constituting the three tasks using the language C.

The three `compile.sh` scripts must compile, respectively, the files `task1.c`, `task2.c`, and `task3.c`, generating, in their own directories, the three respective programs, which must be named: `task1`, `task2`, and `task3`.

## task1

Create a program that takes as input a single command line argument: ⟨`required_address`⟩. If the memory address specified in ⟨`required_address`⟩ is allocated in the virtual memory of your program, your program must print to `stdout` the value of the single byte of memory located at address ⟨`required_address`⟩ and it must exit with exit code `0`. Otherwise, if it is not allocated, your program must not print anything to `stdout`, and it must exit with exit code `1`.

You must not print to `stdout` or `stderr` anything not specified above. Under no circumstances, your program should trigger a `Segmentation fault` exception.

⟨`required_address`⟩ is expressed as a hexadecimal number between `0` and `7fffffffffffff`. I suggest using the function `strtol` to convert ⟨`required_address`⟩ to an `unsigned long` value.

The value of the byte located at the ⟨`required_address`⟩ must always be printed as a two digit, lowercase, hexadecimal number, followed by a new line. To this aim, I suggest using the `printf` function with the format string `"%02x\n"`

To verify if a memory address is allocated, I suggest reading the content of the file `/proc/self/maps`.

While developing and debugging your code, it may be helpful to add a `sleep` function call at the end of your code, so that you can check the content of the file `/proc/`⟨`your_process_pid`⟩`/maps`. This `sleep` function call must be removed in the final version of your code.

For this specific task, you should assume that your code will always be run with memory randomization disabled by using the `setarch` command in the following way:

```
setarch x86_64 -R ./task1 100000
```

For this specific task, your `compile.sh` script must compile your code in the following way:

```
gcc -ggdb -O0 -o task1 task1.c
```

**Examples:**

```
   setarch x86_64 -R ./task1 0
```

It should not print anything to `stdout` and it should exit with exit code `1` (in bash, you can print the exit code of the last terminated command by using the command `echo $?`).

```
   setarch x86_64 -R ./task1 400000
```

It should print to `stdout` the following:

```
7f
```

and exit with exit code `0`.

```
   setarch x86_64 -R ./task1 610000
```

It should print to `stdout` the following:

```
00
```

and exit with exit code `0`.

## task2

Create a program that takes as input a single command line argument: ⟨`assembly_code`⟩. Your program must use the `mmap` function to allocate an executable memory region. Then, it has to hex-decode ⟨`assembly_code`⟩ and copy its hex-decoded content into the allocated executable memory region. Finally, it has to start executing the copied code.

I suggest to use the `sscanf` function with the format string `"%02hhx"` to hex-decode ⟨`assembly_code`⟩. To start executing code at a specific address (in the example below `code_address`), I suggest to use the following code:
`((void (*)(void))code_address)();`

Understanding the assembly code used in the examples below is not required, but you are free to ask me how it works.

**Examples:**

```
./task2 ebfe
```

Your program should enter in an infinite loop and never terminate.

```
./task2 e8200000004889c64831c048ffc04889c74831d2b2040f054831c0b83c0000004831ff0f05e806000000584883c008c3ebf84869210a
```

Your program should print to `stdout` the string (terminated by a new line):
`Hi!`
and exit with exit code `0`.

```
A=$(printf "90%.0s" {1..8000}; printf "4831c0b83c0000004831ff4883c7630f05"); ./task2 "$A"
```

Your program should not print anything to `stdout` nor to `stderr`, but it should exit with exit code `99`. This example assumes that you copied and pasted the line above in a bash shell.

## task3

Create a program that takes as input a single command line argument: ⟨`file_path`⟩. Your program must use the `mmap` function to map the file specified by ⟨`file_path`⟩ into memory. Then, by directly reading from and writing to the allocated memory, it must reverse the content of the given file. The reversed content must be saved into ⟨`file_path`⟩. You must not create any new file.

**Examples:**

You can find the files `testfile1`, `testfile1_reversed`, `testfile2`, and `testfile2_reversed` in the `tar` file of this homework.

```
./task3 testfile1
```

Your program must change `testfile1` so that its content becomes equal to the content of the file `testfile1_reversed`.

```
./task3 testfile2
```

Your program must change `testfile2` so that its content becomes equal to the content of the file `testfile2_reversed`.