

Please read Chapter 2 of the textbook and then answer the following, trying not to look at your notes or at the textbook. Quiz #2, on Thursday 21st September, will consist exclusively of questions taken from the Part I of this homework.

Part I — Short Questions

Question 1

The services and functions provided by an operating system can be divided into two main categories. Describe them, and discuss how they differ.

Question 2

If I want to develop a new application program, should I rather learn the system calls or the API? Explain your answer.

Question 3

What is the role of the system call interface?

Question 4

What does it mean for a process to *lock* shared data? How is it useful?

Question 5

Give three examples of system calls that deal with device management.

Question 6

Name the two common models of communication between processes. If I want to transfer large data between two processes, which one is the fastest?

Question 7

Mechanisms implements *what* to do, or *how* to do it?

Question 8

Can an Operating System be written in more than one programming language?

Question 9

For an operating system to be portable, should it be written in assembly language or in a higher-level language?

Question 10

Should a critical routine like CPU scheduler be written in assembly language, to obtain higher performance?

Question 11

Name an advantage provided by a monolithic structure. Name a disadvantage.

Question 12

Name an example of modular approach.

Question 13

In a layered operating system, does a layer needs to know all the details of the implementation in the lower layers?

Question 14

What is the purpose of a log file?

Question 15

What is System Generation?



Part II — Problems

This week's problems require a computer. They assume you know how to launch your OSC-2016 virtual machine using VirtualBox (or VMware), and that you know how to log-in (remember that the password for the "oscreader" user is "osc" (without the quotes)). I'll assume that you will have successfully completed them by the time Homework #3 is released (Thursday 21st September), so don't wait and let me know if you had difficulties doing them.

Problem 1

Let us start by organizing our folders and files.

1. Create a new folder on your desktop (right-click, and then "New Folder"), name it "hw".
2. Open it (double click on it) and create a new folder in it, that you will name "02".
3. Right-click, and select "Open in terminal". You should see a terminal opening, with

```
oscreader@OSC:~/Desktop/hw$
```

printed. If you are in any other directory (i.e., if, for instance,

```
oscreader@OSC:~/Desktop/hw/02$
```

is printed), then go the right folder by *changing directory*, using the `cd` command:

```
cd ~/Desktop/hw
```

4. First, let's download (*get throug the web*) this homework, using

```
wget -P 02
```

```
↪ http://spots.augusta.edu/caubert/teaching/2017/fall/csci3271/hw/02.pdf
```

5. Now, let's *move* what we did last time using the `mv` command:

```
mv ../hw1 .
```

6. Now, let us *list* the files and folder in the `hw` folder with the `ls` command:

```
ls
```

You should see

```
oscreader@OSC:~/Desktop/hw$ ls
```

```
02      hw1
```

7. Finally, let's *rename* the `hw1` folder by using the `mv` command (indeed, *moving* is taken to be the same as *renaming*):

```
mv hw1 01
```

8. If you use the `ls` command again, you should see

```
oscreader@OSC:~/Desktop/hw$ ls
```

```
01      02
```

Much better!

Problem 2

In this exercise, we will play with one of the implementation of hash functions, the `sha256sum` program.

1. Give an informal description of what a hash function does. Name one of the usage of such a function.
2. Create a folder named `hash` in your `~/Desktop/hw/02/` folder, and enter it, using

```
mkdir ~/Desktop/hw/02/hash && cd ~/Desktop/hw/02/hash
```

Then, create a file named `test1.txt`

```
gedit test1.txt
```

enter “test1” in it, save it, and close gedit. Now, print its SHA256 message digest (its *signature*), using

```
sha256sum test1.txt
```

Can you make any connexion between the content of the file and the signature printed?

3. Make a copy of that file, that you’ll name `test2.txt`, using the `cp` command:

```
cp test1.txt test2.txt
```

Print `test2.txt`’s signature using the `sha256sum` command. Is the signature the same? Can you conclude something regarding the correlation between the name of a file, and its signature?

4. Edit the content of `test2.txt` by changing just one character, and then print its signature. Can you relate that signature to the signature of `test1.txt`? Does changing one character in your file changes only one character in its signature?
5. Define what a *collision attack* is (wikipedia is your friend).

Problem 3

In this exercise, you will be asked to manipulate a simple Makefile and a simple C program. The Makefile file is given in Listing 1, and the `first.c` C program is given in Listing 2.

- (a) *Using the command line*, create a folder named `first_C_program` in your `hw/02` folder, go in it, and download the files hosted at <http://spots.augusta.edu/caubert/teaching/2017/fall/csci3271/hw/02/Makefile> and <http://spots.augusta.edu/caubert/teaching/2017/fall/csci3271/hw/02/first.c>, using

```
wget
```

```
→ http://spots.augusta.edu/caubert/teaching/2017/fall/csci3271/hw/02/Makefile
→ http://spots.augusta.edu/caubert/teaching/2017/fall/csci3271/hw/02/first.c
```

- (b) Inspect the two files. They are simple Makefile and C programs. Notice the similarities, and the differences, between the C programming language and the programming languages you are familiar with. In the future, I will assume that you are familiar with the structure used in those two files.

Problem 4

Read the discussion “How can I copy a file on Unix using C?” at <https://stackoverflow.com/q/2180079> (cached version), and answer the following:

- (a) What is “sparseness” of a file, what is “btrfs cow”?
- (b) What is a `goto` statement?
- (c) What is Lothar, in his/her comment from Jul 17 ’12 at 21:22, complaining about?
- (d) What is “Bobby Tables attack”, and why does “sanitizing” prevent them?

```
# I'm a comment.

# CC is a constant to store the name of the compiler we want to use.
CC=gcc

# This is the "all" rule, the command "make" will execute that rule if no
→ argument is given.
all: first.c
$(CC) first.c -o bin_first

# If we type "make clean", then this command will be executed.
clean:
rm -rf bin_first

# To execute your program, we could add a rule like
run: all
./bin_first
# And it would indeed work: we could just type "make run" to execute our
→ program.
# However, this is considered a bad practice: read and comment on
→ https://stackoverflow.com/a/904011
```

Listing 1: Makefile

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    int counter,
    number_of_throws = 5,
    threshold = 40;

    /* This integer will act as a Boolean, 0 for false, 1 for true. */
    int greater_than_threshold = 0;

    /* A simple array. */
    int values[number_of_throws];

    /* Initializes random number generator. */
    srand(time(NULL));

    /* Store and print number_of_throws random numbers from 0 to 49. */
    for (counter = 0; counter < number_of_throws; counter++) {
        values[counter] = rand() % 50;
        printf("Throw #d: %02d\n", counter + 1, values[counter]);
    }
```

```

        if (values[counter] > threshold) {
            greater_than_threshold = 1;
        }
    }

    /* Simple pointer manipulation. */
    int* p = values;
    int sum = 0;
    for (counter = 0; counter < number_of_throws; counter++) {
        sum += *p;
        p++;
    }
    printf("The sum of your throws is %d.\n", sum);

    /* If ... else statement. */
    if (greater_than_threshold) {
        printf("You had one throw greater than %d\n", threshold);
    }
    else {
        int new_chance;
        /* While loop. */
        while (!greater_than_threshold) {
            new_chance = rand() % 50;
            if (new_chance > threshold) {
                greater_than_threshold = 1;
                printf("You finally had a throw greater than %d,
                    ↪ with %02d!\n", threshold, new_chance);
            }
            else {
                printf("%02d, no luck, let's try again.\n",
                    ↪ new_chance);
            }
        }
    }
    return (0);
}

```

Listing 2: first.c

