Quiz #1, on Wednesday, September 5th, will consist of questions taken from or inspired Parts I and II of this homework.

Part I — Questions

1. An operating system is sometimes called a "resources allocator", give two examples of resources the OS allocates.

2. What is the difference between a single-purpose and a general-purpose processor?

3. If I double the number of processors in my computer, will my computer run exactly two times faster?

4. Explain what graceful degradation is.

5. A multi-processor system where every processor can execute all tasks is called symmetric or asymmetric multiprocessing?

6. Are all multicore system multiprocessor?

7. A program in execution is called a . . . ?

8. What is the name of an interrupt sent by a software?

9. What is a volatile storage device? Name an example of volatile memory.

10. What is direct memory access?

11. What is the storage device that the CPU is able to address and access directly?

12. Which one is usually the fastest, the cache or the main memory?

13. In a multiprocessor environment where each CPU has a local cache, what is cache coherency and why does it matter?

14. What is the difference between protection and security?

15. What is reverse engineering?

16. Expand the acronym **CLI** and **GUI**.

Part II – Problems

I'll assume that you will have successfully completed those tasks by Wednesday, September 5th, so don't wait and let me know if you had difficulties solving them. Problems 1 and 2 are of particular importance, since they help you to set your computer for the rest of the semester.

Problem 1

In this exercise, we will install a software that will allow us to run virtual machines on your computer, and run our first virtual machine. You will need administrator privileges, an Internet connection, and some time.

Below are the instructions to virtualize a Linux 3.1 kernel using Debian (v 8.2) distribution. If you want to use another virtualizer, or another linux-based distribution, feel free to do so, since it should not change much (the image used below contains some source code, but I'll give you access to copy of it if we were to use it). Just make sure you have access to the "usual" compilers and softwares (cf. Problem 2) and make sure you know the basics commands to use a terminal (`cd`, `mkdir`, `ls`).

1. We first install VirtualBox, which is an (almost) open-source and completely free software that allows to run virtual machines. Go to `https://www.virtualbox.org/wiki/Downloads` and click on your operating system below "VirtualBox 5.2.18 platform packages". The steps to install this software from here should be immediate.

2. Download the following file: `http://people.westminstercollege.edu/faculty/ggagne/osc/vm/OSC-2016.ova`. This is the image of a Debian installation with the source files of *Operating System Concepts Essentials, 2nd Edition*.

3. Launch VirtualBox, click on "File", and then on "Import Appliance" (or, preferably, do "Ctrl" + "i"). Select the previously downloaded `ova` file, and click on "Next" and then on "Import".

4. Now you should come back to the first view you had, except that "OSC-2016 – Powered Off" should appear on the left menu on your screen. Click on it, and then on "start".

5. Your virtual machine should start in a new window. VirtualBox will probably inform you that your keyboard and mouse will be "captured" by the virtual machine every time this window is active.

6. Your machine should start and ask for the password of the user "oscreader": enter "osc" (without the quotes).

You're done!

Problem 2

In this exercise, we will make sure that your virtual machine can compile and execute programs from three different programming languages: `C`, `C++` and `java`.

1. First, start your virtual machine: launch VirtualBox, start OSC-2016 and log-in.

2. (Optional: you may want to open firefox and load this homework, located at `http://spots.augusta.edu/caubert/teaching/2018/fall/csci3271/hw/01.pdf`, to relieve you from a part of the typing burden. Be careful when you copy and past, though.)

3. Open a terminal (right-click on your desktop, and click on "Open Terminal", or, alternatively, click on "Activities", and then on "Terminal"). The rest of this exercise will take place in this terminal.

4. Create a folder with

   ```
   mkdir hw1
   ```

   and enter it with

   ```
   cd hw1
   ```

5. We will first compile and execute a `C` program.

   (a) Create a folder and enter it:

```
mkdir C && cd C
```

(b) Launch the default editor and create a new file with the command

```
gedit hello.c
```

In this new window, type the following

```
#include <stdio.h>
int main()
{
        printf("Hello!\n");
        return 0;
}
```

save and quit. You may see that there was an error message printed in the console:

```
** (gedit:23310): WARNING **: Error when getting information for file
 ↪ '/home/oscreader/hw1/c/hello.c': No such file or directory
```

you can safely ignore it.

(c) Compile your program with

```
gcc -o hello hello.c
```

and execute it with

```
./hello
```

(d) Your program should print "Hello!" and a new line in your terminal. Return to the parent folder with

```
cd ../
```

6. We will now compile and execute a C++ program.

(a) Create a folder and enter it:

```
mkdir Cplusplus && cd Cplusplus
```

(b) Launch the default editor and create a new file with the command

```
gedit bonjour.cpp
```

In this new window, type the following

```
#include <iostream>
int main()
{
        std::cout << "Bonjour!\n";
        return 0;
}
```

save and quit. You may see an error message printed in the console, you can safely ignore it.

(c) Compile your program with

```
g++ -o bonjour boujour.cpp
```

and execute it with

```
./bonjour
```

(d) Your program should print "Bonjour!" and a new line in your terminal. Return to the parent folder with

```
cd ../
```

7. To conclude, we will compile and execute a `java` program.

(a) Create a folder and enter it:

```
mkdir java && cd java
```

(b) Launch the default editor and create a new file with the command

```
gedit Hallo.java
```

In this new window, type the following

```
public class Hallo
{
        public static void main(String[] args)
        {
                System.out.print("Hallo!\n");
        }
}
```

save and quit. You can see that there was an error message printed in the console, you can safely ignore it.

(c) Compile your program with

```
javac Hallo.java
```

and execute it with

```
java Hallo
```

(d) Your program should print "Hallo!" and a new line in your terminal. Return to the parent folder with

```
cd ../
```

I will now assume that you know how to create, compile and execute a simple program with those three languages. If a program were to freeze or loop, use "Ctrl" + "c" to stop it when the focus is on the terminal.

Problem 3

Since there is no "official textbook" for this class, but only a list a recommendation (cf. `http://spots.augusta.edu/caubert/teaching/2018/fall/csci3271/#textbook`), I will try to give you some articles and passages from various source to read from time to time. Let us start by a possible answer to a question you maybe asked yourself, "Why Take an operating systems Course?": `https://blog.regehr.org/archives/164`

Problem 4

This last problem is strongly inspired by the "Introduction to operating systems" of *operating systems: Three Easy Pieces*, available at `http://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf`.

Compile the following code, using for instance

```
gcc -o cpu cpu.c
```

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

int main(int argc, char* argv[])
{
        char* str = argv[1];
        int i = 0;
        while (i < 5) {
                sleep(1);
                printf("%s\n", str);
```

```
            i++;
        }
        return 0;
}
```

Now, execute it with

```
./cpu "A"
```

Is the behaviour of the program consistent with your understanding of the code? It's all right if you don't understand it fully, just make sure you understand what the body of the `while` loop is doing.

Now, we can run multiple instances of this program (to which we gave different arguments) using

```
./cpu "A" & ./cpu "B" & ./cpu "C" & ./cpu "D"
```

You could suppose that "A" would be printed 5 times, then "B" would be printed 5 times, then "C" and "D": is that what happened?

Increase the value in the test of the loop (from 5 to e.g. 10), compile the program, and run multiple instances of the program at the same time: is the order in which the programs are executed always the same (i.e., is it always "A B C D", *in that order*) that is displayed at the screen?

Quoting the Section 2.1, "Virtualizing the CPU" of the Introduction to operating systems:

> Well, now things are getting a little more interesting. Even though we have only one processor, somehow all four of these programs seem to be running at the same time! How does this magic happen?
>
> It turns out that the operating system, with some help from the hardware, is in charge of this **illusion**, i.e., the illusion that the system has a very large number of virtual CPUs. Turning a single CPU (or small set of them) into a seemingly infinite number of CPUs and thus allowing many programs to seemingly run at once is what we call **virtualizing the CPU** [...].
>
> You might also notice that the ability to run multiple programs at once raises all sorts of new questions. For example, if two programs want to run at a particular time, which *should* run? This question is answered by a **policy** of the OS; policies are used in many different places within an OS to answer these types of questions, and thus we will study them as we learn about the basic **mechanisms** that operating systems implement (such as the ability to run multiple programs at once). Hence the role of the OS as a **resource manager**.

Note that shells include mechanisms to force the operating system to have a sequential processing of the programs: if you replace the `&` with `&&` and run

```
./cpu "A" && ./cpu "B" && ./cpu "C" && ./cpu "D"
```

then you will notice that `./cpu "A"` complete before `./cpu "B"` starts. This can be useful if you need to impose some order in which multiple programs needs to be executed.