System and device programming
20 July 2011

(Theory: no textbooks and/or course material allowed)

(15 marks) The final mark is the sum of the 1st > 8 and the 2nd part > 10

The final mark cannot be refused, it will be registered (no retry for marks >= 18)

1. (3.0 marks) Show and describe the fields of a Unix Inode. It is possible that a 1 Mbytes file occupies less than 256 blocks of 4 Kbytes? If yes, how the file has been created, and what does each block contain?

2. (4.0 marks) Complete the scheme of two **parallel cyclic threads**, inserting the appropriate semaphore calls and variable management that allow printing either the word **ABCD** or **BACD**.

```
P1:  while (TRUE){
       ...
       printf("A");
       ...
       printf("C");
       ...
}
```

```
P2:  while(TRUE){
       ...
       printf("B");
       ...
       printf("D\n");
       ...
}
```

3. (3.0 marks) Write the sequence of **Unix** system calls that allow a process to change its code**,** using a system call of the **`exec`** family, to execute the command **`"ls -l"`** redirecting its output to a **temporary file** (not seen by the other processes)**`output.txt`**.

4. (2.0 marks) Explain the main features of memory mapped files in Win32. In particular, list the main functions (or operations) that can be used to work on memory mapped files and their differences with respect to sequential and direct file access routines.

5. (3.0 marks) Describe the main features of standard IO devices and console in Win32. Is it possible to write a function **`PrintMsg`** that, after receiving as parameters a handle to a file and a message, exploits different output functions for standard files and for the console? If it is possible, write an example in C language.

System and device programming
20 July 2011

(C program: textbooks and/or course material allowed)

(18 marks) The final mark is the sum of the 1st > 8 and the 2nd part > 10
The final mark cannot be refused, it will be registered (no retry for marks >= 18)

Write a C program in the **Win32** environment which works as a buffer between a sets of sensors and a set of log files.

The main program receives two arguments on the command line: an input file name, and M.

The input file includes **fixed length records**, each one containing two filenames (as text strings, standard or UNICODE chars upon choice of the student). The first filename specifies the file which includes the input data of a sensor. The second filename is the name of the output log file. Notice that the same filename may appear on multiple lines of the file. The following is a correct example of the input file:

```
sensor001.dat output001.dat
sensor002.dat output001.dat
sensor001.dat output002.dat
sensor004.dat output003.dat
...
```

The main program starts M threads. Each thread hat to read lines from the input file, and for each line it must:

- Read the sensor file, containing data from the sensor as a sequence of 8 byte floating point numbers that represent a sequence of readings from the related sensor.
- Compute, from the data sequence, the total number of data, the average, the maximum and the minimum values.
- Write a text line in the corresponding log file.

All operations need to be done avoiding read/write conflicts, and maximizing concurrent processing. Access to the common input file (the one specified on the command line) must be done concurrently using file locking. The lines in the log files need to be consistent, i.e., there should not be broken and/or mixed lines, containing data statistics from different sensors.