COMPUTER ARCHITECTURE AND OPERATING SYSTEMS (CAOS)
DAC, FIB, UPC
2024/2025 – Spring term

Final exam **Tue 10 June**

- This exam is **open book**, **closed Internet**.
- Write your answers, in order, in a text file named `<your_name>.txt`
- Time **120 minutes**

1. (4 points) Short questions.
   (a) Bit is a blend word, a combination of parts of other words or sounds. List what they are.

   Binary Digit

   (b) Of the phases of an instruction cycle, list and explain three that always occur.

   - Fetch: The CPU retrieves the instruction from memory (or cache).
   - Decode: The fetched instruction is interpreted by the control unit.
   - Execution: The CPU performs the operation specified by the instruction.

   (c) List and explain the function of two CPU registers that are involved in the execution of every instruction.

   - Program Counter (PC) / Instruction Pointer (IP): Holds the memory address of the next instruction to be fetched.
   - Instruction Register (IR): Stores the currently fetched instruction while it is being decoded and executed.

   (d) What differentiates the operating system from any other programme installed on your computer?

   The operating system (OS) is fundamentally different from other programs installed on a computer due to its core responsibilities, privileged control over hardware, and role as an intermediary between users/applications and the computer's resources.

   (e) Of the three situations that cause the operating system to take control of the CPU, in which of them can the OS control the exact time at which it will start?

   The operating system (OS) can take control of the CPU in three primary situations: system calls, exceptions and hardware interrupts. However, only the timer interrupt is triggered by hardware timer configured by the OS. The OS controls the interval between interrupts (e.g., 10ms time slices) and when timer interrupts are enabled/disabled.

   (f) Explain the differences between a programme and a process. Can one process create several programs? And conversely, a program can create several processes?

   Program = Static code on disk. Process = Running instance of a program. A process can create programs (if it explicitly generates executable files). A program can create multiple processes (common in multitasking systems).

(g) How many file descriptor tables are there in memory? and File Tables? and Inode tables? Write a Linux command line that displays the contents of the file descriptor table of process 1234.

- File Descriptor Tables: One per process (e.g., thousands if many processes run).
- Open File Table: One system-wide table (shared across all processes).
- Inode Table: One system-wide table (managed by the filesystem).
- `ls -l /proc/1234/fd/`

(h) Explain the difference between page and frame. Can the logical space of a process be larger than the physical space of the system, and conversely, can there be more physical space in the system than the logical space of any process?

- A Page is a fixed-size block of logical memory (in virtual address space). Used by the process (visible to software). A Frame is a fixed-size block of physical memory (in RAM). Used by the OS/hardware (actual storage location). The OS maps pages (virtual) to frames (physical) via the page table. Pages and frames are typically the same size (e.g., 4KB in x86).
- Yes. Virtual memory allows larger logical space than physical RAM (via paging/swap).
- Yes. Physical memory can exceed a process's limits but is shared system-wide.

2. (2 points) Floating point representation. Suposse we are using the standard IEEE 754, 6b, normalized. We consider a 6-bit format:

- There is one sign bit s.
- There are $k = 2$ exponent bits. The bias is $2^{k-1} - 1 = 1$.
- There are $n = 3$ fraction bits.

For example, in this format, the codification in bits of the decimal one is 001000. Write down the binary representation for the numbers three and 15/8. Show all the calculation steps.

Value	Bits
one	`0 01 000`
three	`0 10 100`
15/8	`0 01 111`

three: 0 10 100

The decimal 3.0 in binary is 11.0 (since $3 = 2^1 + 2^0$). This can be written in normalized form as: $1.10 \cdot 2^1$

Sign bit (s): 3.0 is positive $\rightarrow s = 0$.

Exponent (exp): The exponent term is 2^1, so E = 1. Since E = exp - bias and bias = 1, we have: exp = E + bias = 1 + 1 = 2. In 2-bit binary, 2 is 10.

Fraction (frac): The significand is 1.10 (from 1.102^1). The fractional part is 10, so mantissa = 100 (we take the first 3 bits, padding with a 0 if necessary, but here 10 becomes 100 because we need 3 bits).

Thus, the 6-bit representation is **010100**

15/8: 0 01 111

1.875 in decimal. Binary representation of 1.875: Integer part (1) $\rightarrow$ 1 in binary.

Fractional part, 0.875 $\rightarrow$ 0.111 in binary (since $0.875 = 0.5 + 0.25 + 0.125$). Combined: 1.111 in binary. The normalized form is already $1.111 \cdot 2^0$ (since the binary point is after

the first 1).

Sign bit (s): 15/8 is positive $\rightarrow$ $s = 0$.

Exponent (exp): The exponent term is 2^0, so E = 0. Since E = exp - bias and bias = 1, we have: exp = E + bias = 0 + 1 = 1. In 2-bit binary, 1 is 01.

Thus, the 6-bit representation is **001111**

3. (3 points) Interprocess communication. Read the Python code in listing 1 carefully and answer the questions. Error checking has been omitted for readability. Assume it runs without error.

Listing 1: "nlswc.py"

```
1  import os
2  import sys
3  import signal as s
4  pid = [0,0]
5  def timeout(signo,frame):
6      global pid
7      os.kill(pid[0],9)
8      os.kill(pid[1],9)
9      return 0
10 s.signal(s.SIGALRM, timeout)
11 s.alarm(3)
12 fd = os.pipe()
13 pid[0] = os.fork()
14 if ( pid[0] == 0):
15     os.close(fd[0])
16     os.close(1)
17     os.dup(fd[1])
18     os.set_inheritable(1, True)
19     os.execlp("ls","ls","-gRitalo","/")
20     sys.exit(1)
21 else:
22     pid[1] = os.fork()
23     if (pid[1] == 0):
24         os.close(fd[1])
25         os.close(0)
26         os.dup(fd[0])
27         os.set_inheritable(0, True)
28         os.execlp("wc","wc")
29         sys.exit(2)
30     else:
31         os.close(fd[0])
32         os.close(fd[1])
33 os.wait()
34 os.wait()
35 print(s.alarm(0))
```

(a) At most, how many processes are running concurrently? How many are parent processes? How many are child processes?

3 processes, 1 parent process and 2 children processes at most.

(b) Write down the file descriptor table of the "`ls`" process just after executing the call to `os.dup(fd[1])` (line 17).

Since the first child has not closed the write pipe after duplicating it, just after calling dup it will have:

```
0 -> /dev/pts/1
1 -> 'pipe:[220334]'
2 -> /dev/pts/1
```

```
4 -> 'pipe:[220334]'
```

(c) What signals can these processes receive related to the pipe they are sharing?

- SIGPIPE to the process that try to write to a pipe without readers

(d) Explain what this program writes to standard output and why.

The program executes an `ls|wc`. If this execution lasts less than three seconds, the output will be three numbers, written by the second child (`wc`) and one number written by the parent (the number of seconds left before the 3 seconds are up). But if it lasts longer than 3 seconds, a `SIGALRM` will arrive at the parent process and it will kill its children. The output in this case will be a number (0) written by the parent and whatever the second child has time to write (possibly nothing).

(e) From shell, what command line do you have to execute to send a SIGHUP to the parent process?

ps To find out the pid of the parent process: `ps -ef | grep nlswc`
kill To send a signal: `kill -HUP <parent process id>`

(f) What signals cannot be caught, blocked, or ignored?

`SIGSTOP, SIGKILL`

4. (1 point) Input/Output. Write a program, in Python or C, named **insertarX** that inserts in a text file (argument 2) a letter (argument 1) between the two last characters. The programme has to be generic, independently of the size of the input file it will always write the letter between the two last characters. You can only use the system calls seen in class. Example of execution:

```
$ echo -n "CTGA" > input.txt
$ ./insertaX T input.txt
$ cat input.txt
CTGTA
```

Error checking has been omitted for readability.

```
1        letter = sys.argv[1]
2        filename = sys.argv[2]
3        fd = os.open(filename, os.O_RDWR)
4        os.lseek(fd, -1, os.SEEK_END)
5        last_char = os.read(fd, 1)
6        os.lseek(fd, -1, os.SEEK_END)
7        os.write(fd, letter.encode() + last_char)
8        os.close(fd)
```