QUIZ-02
- K22 – V. Atlidakis
Name and student id _____

1) [**12 points**: +2 pt on each correct answer, -2 on each incorrect answer]
   - **Choose synchronous or asynchronous.**
     - System call: **synchronous** / asynchronous
     - Exception of type "abort": **synchronous** / asynchronous
     - Exception of type "fault": **synchronous** / asynchronous
     - Exception of type "trap": **synchronous** / asynchronous
     - Procedure call: **synchronous** / asynchronous
     - Timer interrupt: synchronous / **asynchronous**

2) [**20 points**: +2 pt on each correct answer, -2 on each incorrect answer]
   - **Choose Yes or No.**
     - The bootloader is in ROM: Yes / **No**
     - The stack is a static segment: Yes / **No**
     - A thread is a program in memory: Yes / **No**
     - POSIX was invented for portability: **Yes** / No
     - A timer helps implement fault isolation: Yes / **No**
     - A process is used to execute a program: **Yes** / No
     - A mode register helps implement preemption: Yes / **No**
     - A user space application can directly call a syscall: **Yes** / No
     - On a TLB miss, a main memory access must follow: **Yes** / No
     - On x86, a double-fault is an exception of the type "fault": Yes / **No**

3) **[28 points]** Static and dynamic linking.

(a) [10 points] Describe **two disadvantages** of static linking.
- Large binaries in storage.
- Duplication of common code in main memory.
- Slower startup because of potentially unnecessary work that could have been deferred for later.
- Someone wrote: "more difficult to update because it requires recompilation."—Valid point.
- Slower compilation time that can be avoided.

(b) [10 points] Describe **one advantage** and **one disadvantage** of dynamic linking.
- Advantage: Deduplication of common code in main memory.
- Advantage: Faster startup because of the potential to defer unnecessary work for later.
- Disadvantage: Runtime dependencies: a massive problem in reality.

(c) [8 points] If you have a small instruction TLB (i-TLB) but **wish to maintain a high hit rate**, would you prefer **static or dynamic linking?** Explain your choice.
- Static — the routines of the library that are used by the program are packed together in the pages of the code segment. This has a considerably better probability of leading to a good TLB hit rate because of better spatial locality. On the other hand, with dynamically linking, the code segment of the library (including all the routines, both used and not used by the program) is mmap'ed in the process. Therefore, the most likely result is to end up touching more pages, since the in-use routines will be spread out among more pages.

4) **[20 points]** The first prompt below shows the source code of a file named main.c and the second prompt shows an execution of the respective executable. Assume 0x**foo** is a hexadecimal virtual address, and the identifier of the process executing "main" is 1000. Finally, prompt-3 shows a subset of the virtual memory address range mappings of process 1000 executing "main".

[prompt-1]: cat main.c

```
#include <stdio.h>

const char *message = "Hello, World!\n";

void main(void) {
        printf("%p: %s\n", &message, message);
}
```

[prompt-2]: gcc -o main main.c
[prompt-2]: ./main
0x**foo**: Hello, World!

[prompt-3]: cat /proc/1000/maps

```
a) aaaae2de0000-aaaae2de1000   r-xp   … /home/parallels/main
b) aaaae2df0000-aaaae2df1000   r--p   … /home/parallels/main
c) aaaae2df1000-aaaae2df2000   rw-p   … /home/parallels/main
d) fffff6426000-fffff6447000   rw-p   … [stack]
```

---

a) [5 points] 0x**foo** would normally belong in which of (a), (b), (c), (d), or (none)?

Fill in here: (C)

b) [15 points] Explain your choice.

- Observe that we are printing the address of the pointer, and not the address where the pointer points to.
- Most people got confused and answered (b), due to the fact that the content where the pointer points to is indeed intended to be immutable. But we are printing the address of the pointer!
- Obviously, also, the choice cannot be (a), because this VMA range contains executable instructions.
- Finally, it cannot be (d), which is the stack—at the bottom of the Virtual Address Space (VAS)
- Initialized global/static variable ⇒ .data [see page 9, lecture-04]

5) **[20 points]** There is a processor named **di-reverted** with a pipeline with the following five stages:
"IF: Instruction Fetch" → "ID: Instruction decode" → "EX: Execute" → "WB: Register Write Back" → "MEM: Mem. Access".

Observe that the last two stages in the pipeline of **di-reverted** are reverted compared to what we discussed in class!

The processor **di-reverted** supports an instruction called **k22inc**, with syntax **k22inc <%reg> <const>**, which first adds the value **4** to register **%reg** and then adds the constant **const** to the memory location pointed by the updated value of register **%reg**.

The semantics of **k22inc** in C code would look like:

- reg += 4;
- mem[reg] += const;

Reverting the last two stages of the pipeline compared to what we have discussed in class was never a good idea.

a) [10 points] **Specifically for supporting virtual memory**, why is the instruction **k22inc** problematic?
   - A memory translation (from virtual to physical) for mem[reg] is required. Therefore, the potential for an address translation fault exists. On an address translation fault (**remember: fault ⇒ restart the faulting instruction)**, the OS will intervene, **save the current execution state** (i.e., the updated register %reg), invoke the appropriate exception handler to handle the fault, and finally, the "faulting" instruction will be restarted. Unfortunately, the state stored by the exception handler has the register %reg already updated with +4, and the restart will add +4 again. Thus, the instruction k22inc may point to an unintended memory address, depending on whether an address translation fault will be raised or not.

Assume **di-reverted** also supports an instruction **k22help,** with syntax **k22help <%reg> <const>,** which just "touches" (i.e., accesses) the memory at address “reg + const” and does nothing else.

The semantics of **k22help** in C code would look like:

- (void) mem[reg+const];

b) [5 points] Explain how you could use the instruction **k22help** to help with the problem identified in (a)?
   - Add **k22help** before any invocation of **k22inc** to prevent the possibility of an address translation fault.
   - The pipeline of di-reverted is indeed generally problematic. However, specifically w.r.t. virtual memory, anyone who sees how k22help can be used to prevent an address translation fault gets full credit.

c) [5 points] Given your modification suggested in (b) to use the command **k22help**, would you prefer your system to have preemption on or off? And why?
   - Off.
   - Any solution presented in (b) can be undone with preemption on.
   - If the process executing the program containing the pair of the two instructions gets scheduled out at the end of the first instruction (k22help), when the process gets scheduled in again, it is unclear whether it will get an address translation fault or not.