Homework 5

Problem 1

Please specify **which kind of exception** (Faults, Aborts, Traps, Interrupts) will occur in the given scenario, point out whether it is **asynchronous** or **synchronous**, and specify **where the exception handler will return to**.

A. Access content at address 0x0.

Faults. Synchronous. Aborts the interrupted program.

B. The memory of your PC corrupted.

Aborts. Synchronous. Aborts the interrupted program.

C. You run a command "kill -9 <pid>" in your shell.

Traps (System calls). Synchronous. Return control to the next instruction.

D. You click your mouse.

Interrupts. Asynchronous. Return control to the next instruction.

Problem 2

Consider three processes with the following starting and ending times:

Process	Start time	End time
A	0	2
B	1	4
C	3	5

A. For each pair of processes (AB, AC, BC), indicate whether they run concurrently or not.

AB: concurrent; AC: not concurrent; BC: concurrent.

B. Suppose the three processes are running on a single-core machine. Process A is running a program with an infinite loop, as is shown below:

```
int main(void) {
    while (1) {
        /* waste CPU time */
    }
}
```

Would this process A block the execution of process B? Why?

No. Process A is preempted after it uses the processor for a time slice, then context switch happens and process B or other processes can execute on this processor.

Problem 3

A. A process in user mode is not allowed to execute privileged instructions such as those intend to initiate an I/O operation. How does a C program manage to print characters on your screen when it invokes ‘printf’ function in user mode?

   ‘printf’ function works by invoking libc functions which issue system calls (in this case, ‘write’ system call). System call handlers execute in kernel mode, thus is able to initiate I/O operations.

B. Operating system kernel lives in the top portion of every process’s private address space, inaccessible to processes in user mode. Can you give some reasons of why not let kernel live in its separate address space for protection? (Hint: what might be different for system calls?)

   If kernel lives in a separate address space, then each kernel/user mode switch involves a costly context switch. System calls have to be issued by inter-process-calls with parameters passed in messages instead of pushing them to stack directly (keep in mind that different address space means stack content not shared). Cases where system call arguments are pointers should be carefully dealt with.