# 1 Practice Questions

## 1.1 Processes

List the two important illusions that the process abstraction provides to programs. For each illusion, list a mechanism involved in its implementation.

## 1.2 Virtual Memory

One purpose of virtual memory is to allow programs to use more memory than is available in the physical memory by storing some parts on disk transparently. Name some *other* useful thing that can be done with the virtual memory system.

## 1.3 TLBs

Does a TLB (Translation Lookaside Buffer) miss always lead to a page fault? Why or why not?

## 1.4 Java and C

Name some differences between Java and C.

## 1.5 Stacks and Structs

Let's look at a program program which includes the definition for a data structure type:

```
typedef struct data_struct
{
int a;
int *b;
int c ;
} data_struct;
```

This is a small snippet of code for a function foo, which has just been called and in turn calls `print_struct`:

```
int foo() {
 data_struct x;
 int n = 13;
 x.a = ???;
 x.b = &n;
 x.c = 3;
 print_struct(&x);
}
```

Definition of the `print_struct` function:

```
void print_struct(data_struct *y) {
 printf("%p\n", y);
 printf("%d\n", *(y->b + y->c));
<<execution is suspended here>>
}
```

Execution is suspended after the `printf` statements in `print_struct` but before it returns to foo. The stack at this point of the execution of the program is shown below in 4-byte blocks. Note that the stack is shown as is tradition, from bottom to top, with the top-most of the stack at the bottom or lowest address:

```
0x7fffffffffffa040:    0x00203748
0x7fffffffffffa03c:    0x00000001
0x7fffffffffffa038:    0x0000015f
0x7fffffffffffa034:    0x00000000
0x7fffffffffffa030:    0x00402741
0x7fffffffffffa02c:    0x00000000
0x7fffffffffffa028:    0x00000003
0x7fffffffffffa024:    0x7fffffff
0x7fffffffffffa020:    0xffffa014
0x7fffffffffffa01c:    0x00000000
0x7fffffffffffa018:    0x00000007
0x7fffffffffffa014:    0x0000000d
0x7fffffffffffa010:    0x00000000
0x7fffffffffffa00c:    0x00402053
```

1. What is the value stored in the stack at the 8 bytes starting at location `0x7fffffffffffa00c` to `0x7fffffffffffa013` and what does it represent?

2. What value was assigned to `x.a` in the function `foo` and at what address is it stored on the stack?

# 2 Solutions

## 2.1 Processes

1. Logical control flow: the process executes as if it has complete control over the CPU. The OS implements this by interleaving execution of different processes via context-switching(exceptional control flow...).

2. Private linear address space: the process executes as if it has access to a private contiguous memory the size of the virtual address space.

## 2.2 Virtual Memory

1. Sharing of a single physical page in multiple virtual address spaces (e.g., shared library code).

2. Memory protection mechanisms (e.g., page-granular read/write/execute permissions or protecting one processs memory from another).

## 2.3 TLBs

No. The TLB caches page table entries. After a TLB miss, we do an in-memory page table lookup. A page fault occurs if the page table entry is invalid.

## 2.4 Some Differences between Java and C

1. C allows pointer arithmetic; Java does not.

2. C pointers may point anywhere (including the middles of memory objects); Java references point only to the start of objects.

3. C pointers may be cast arbitrarily (even to non-pointer types); casts of Java references are checked to make sure they are type-safe.

## 2.5 Stacks and Structs Solution[1]

```
0x7fffffffffffa040:    0x00203748
0x7fffffffffffa03c:    0x00000001
0x7fffffffffffa038:    0x0000015f
0x7fffffffffffa034:    0x00000000
0x7fffffffffffa030:    0x00402741
0x7fffffffffffa02c:    0x00000000 << padding (external fragmentation), offset +20
0x7fffffffffffa028:    0x00000003 << x.c, offset +16
0x7fffffffffffa024:    0x7fffffff << high order bytes of x.b
0x7fffffffffffa020:    0xffffa014 << low order bytes of x.b, offset +8
0x7fffffffffffa01c:    0x00000000 << padding (internal fragmentation)
0x7fffffffffffa018:    0x00000007 << x.a, offset +0
0x7fffffffffffa014:    0x0000000d << int n = 13
0x7fffffffffffa010:    0x00000000 << high order bytes of return address from print_struct
0x7fffffffffffa00c:    0x00402053 << low order bytes of return address from print_struct
```

[1] Erratum: In actuality we would expect the return address to be 8-byte aligned, but here it is not (`0x7fffffffffffa00c`)