COMP 3659 – Operating Systems
Memory Management Lab, Part 2: More Paging

Instructions:

- Connect your lab workstation to the MACO VPN, and then log into your personal Linux VM.
- Working at separate lab workstations (physically distanced), each student must complete each lab activity below. However, students are encouraged to form learning partnership groups of up to three members. Group members should communicate as they work and should help each other as they engage with and make sense of each activity. This can include:
    - Developing a common understanding of the purpose of each activity;
    - Articulating uncertainty and questions;
    - Sharing insights and suggestions for additional experimentation and exploration;
    - Helping each other to understand, explain, and make sense of observed phenomena, and to make predictions about phenomena;
    - Articulating and summarizing what you have discovered or learned; and
    - Helping each other with coding and Linux-related tasks, including helping each other when stuck.
- Complete each of the following lab activities.

Lab Activities:

Note: This lab assumes that you have already created a `labs` directory directly under your home directory. The path names provided here correspond to this arrangement.

Your instructor is proving you with source code to be used in the following lab activities. Create a copy of this source code for yourself:

```
cp –r /library/comp3659/labs/paging ~/labs
```

Finally, change into:

```
cd ~/labs/paging
```

1. Your task in this lab is to complete a program that simulates the logical-to-physical address translation operation of a MMU in a hypothetical paged-memory environment.

   To begin, assume the following architectural parameters:

   - Byte-addressable memory
   - 4k page size
   - Each logical address is a 15-bit number
   - Each physical address is a 24-bit number

As a warm-up, answer the following questions. Draw pictures and diagrams to assist as necessary:

- Draw a diagram that illustrates the format of a logical address (indicate the page number bits and the offset bits)
- Draw a diagram that illustrates the format of a physical address
- How many bytes are there per page?
- How many pages are there in each process's logical address space?
- What is the total size in bytes of each process's logical address space?
- How many bytes are there per frame?
- How many frames are there in physical memory?
- What is the total size in bytes of physical memory?

2. Assume that process P_0 has the following page table (all numbers shown in hex):

0	3AD
1	F90
2	273
3	11E
4	F00
5	BA1
6	000
7	FFF

As a second warm-up, to what physical address is each of the following logical addresses mapped?

- `0000`
- `2CAD`
- `7FFF`
- `8000` ← trick question!

3. Open `mmu.c` and study its source code. This program, once completed, is intended to simulate the address-translation operation of the MMU for this system.

   Complete the implementation of the `log_to_phys` function such that it simulates the MMU operation correctly for the given page table[1].

   For this step and the next one, ignore the `fault` output parameter (leave it set to 0).

4. Update the solution so that it shows how the given logical address would be translated for both P_0 and a second process, P_1. For the latter, create a second page table of your choice.

[1] Recall C's bitwise and shift operators, including: `&`, `|`, `^`, `~`, `>>`, `<<`. Some of these will be useful here, sometimes in conjunction with masks.

5. Not every process will necessarily make use of its entire logical address space. In that case, it is wasteful to allocate physical frames to pages when they will never be used. One solution is to utilize one of the (currently unused) bits in each page table entry as a "valid/invalid" flag:

   valid bit = 1 page is used by process and is mapped to a frame (frame bits of this entry valid)
   valid bit = 0 page is unused by process and is not mapped to a frame

   Using this approach, modify the page table for P_0 such that it uses only pages 0 through 4. When an unused logical address is supplied to the MMU simulator, it should respond by setting the `fault` flag. In this case, the return value (corresponding physical address) is undefined.