Memory Management- Paging

Assume a physical memory of a given size (8 GB, 16 GB, …) and we wish to manage it, i.e., we need to devise the following algorithms :

1. Allocate
2. Release
3. Read
4. Write

Our Memory Model

Memory is simply a table of words, the latter being the smallest unit that can be allocated, released, read and written.

Each word has a **unique address**. Since addresses depend physically on the size of the address bus, the maximal size of a memory on a 32 bit system is 2^32 words.

1. Since we'll be using a logical memory to enforce memory seperation between executions, each process is going to own its own logical space (we'll be calling it too logical memory, virtual memory). This space by definition is imaginary and does not exist except in the mind of the programmer.

2. All processes are going to share the physical memory though.

Allocation Algorithm

To start off, any use of memory should always start first by allocating the needed space.

To illustrate how it works, we'll use the following code snippet.

```
ptr = allocate(5)

ptr[0] = 55

ptr[4] = 66
```

Let’s implement our memory algorithms in order to make this code work.

Version 1

```
Allocate(nb of words) {
    1. Find nb contiguous words in logical memory
        a. update the free list of logical words accordingly to book them

    2. Find nb words (not necessarily contiguous) in physical memory

        a. update the free list of physical words accordingly to book them
        b. update the mapping table to link a logical word to a physical one

    3. Return the base address of what has been allocated in step 1
}
```

The mapping table could be as follows :

54
55
67
69
13

Address translation is straightforward in this case when writing to ptr[0] and ptr[4].

Having done this, let’s calculate the size of the mapping table.

Size = nb of entries * size of an entry

= 2^32 * (size of an address = 4 bytes)

= 16 GB

Each and every process needs this table to be able to use a logical memory ! Therefore, we need to find a way to reduce its size. The only thing that can be reduced significantly is the number of entries, threfore, instead of mapping a word onto a word, we'll be mapping blocks of words in logical memory (called pages) onto block of words in physical memory (called frames).

The page/frame size in our illustration is 8 Kwords (2^13 words).

Version 2

```
Allocate(nb of words) {
    1. Find nb contiguous words in logical memory
        a. update the free list of logical words accordingly to book them

    2. For each page used in 1, find a free frame in physical memory

        a. update the free list of physical frames accordingly
        b. update the mapping table (page table) to link a page to a frame

    3. Return the base address of what has been allocated in step 1
}
```

The page table after execution of this algorithm could be as follows :

567
Invalid
Invalid
Invalid

Address translation using this version of the allocate algorithm is a bit tricky.

Logical addresses can be writtten as a pair (page number, offset inside the page)

Physical addresses can be written as a pair (frame number, offset inside the frame)

Since a given frame is simply a copy/duplicate of its corresponding page, this implies that the offset inside a page and a frame is the same.

As for the page number, it is used as an index in the page table which leads to the following translations in our case :

ptr[0] with a logical address of (0, 0) has its physical counterpart at (567, 0)

ptr[4] with a logical address of (0, 4) has its physical counterpart at (567, 4)

What if we next consider the following :

```
ptr1 = allocate (2^14)

ptr1[0] = 77

ptr1[2^13+2] = 88
```

Apply it to make sure you’ve understood the whole process !