CSSE 332 -- Operating Systems

Introduction to Memory Virtualization

Name: Solution Key

Question 1. (5 points) Consider a system where each process is mapped directly into memory. In other words, the process can directly access memory without intervention from the operating system.

What are some of the main challenges with this approach?

Solution: Here are a few:

- *Privacy*: How can we prevent a process from accessing things that it has no control over (e.g., the kernel's memory, other processes' memory, etc.)
- *Memory limits*: How can we know how much memory to allocate for a given process? We can't really predict how much memory a process will need.
- *Portability*: Address will change each time the process is loaded into memory. How can we make sure our code is portable without any form of translation?

Question 2. (5 points) In your own words, describe what it means for a process to have *virtual addresses*?

Solution: A virtual address is a fake address that is specific to each process. It is the job of the operating system with help from the hardware to translate the virtual addresses into physical addresses.

Question 3. (5 points) Address translation is the process by which the operating system (and the hardware) translates a **virtual address** into a **physical address**.

Question 4. The questions below refer to the *base and bounds* memory translation approach.

(a) (5 points) Assume that process P_1 gets assigned a base register `base_reg`. Write down the formula used to calculate the *physical address* (`PA`) from a given *virtual address* (`VA`).

Solution:

$$\texttt{PA} = \texttt{VA} + \texttt{base\_reg} \tag{1}$$

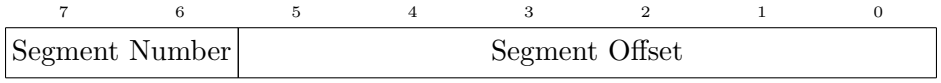
(b) (5 points) Assume that process P_1 gets assigned a base register `0x0048`. When P_1 attempts to access address `0xff04`, which physical address does it end up accessing?

Solution:

$$\texttt{0x0048} + \texttt{0xff04} = \texttt{0xff4c}$$

Question 5. Assume we are running on an 8-bit architecture and we would like to implement memory segmentation. Each process should have the generic four sections: code, globals, stack, and heap.

(a) (5 points) Describe how an 8-bit address would be divided up to perform address translation. You may use the bit-box below.



(b) Assume now that when process P_1 is loaded into memory, it is assignmed the following segment table.

Segment	Base	Bounds	Growth
Code	0x40	0x0f	+
Globals	0x50	0x0A	+
Heap	0x60	0x10	+
Stack	0x7f	0x10	-

i. (5 points) Write down the formula used to translate a virtual address into a physical address using the segment table above.

Solution:

$$\texttt{PA} = \texttt{SegTable}[\texttt{VA}[7:6]] + \texttt{VA}[5:0]$$

ii. (5 points) Assume P_1 attempts to access the virtual address `0x04`, what would be the corresponding physical address? (Write segmentation fault if the access is invalid).

Solution:

- Segment number is `00`, which means base is `0x40`.
- Offset is `0x04` (watch for it being only 6 bits)

So $\texttt{PA} = \texttt{0x40} + \texttt{0x04} = \texttt{0x44} < \texttt{0x40} + \texttt{0x0f}$

iii. (5 points) Assume P_1 attempts to access the virtual address `0x84`, what would be the corresponding physical address? (Write segmentation fault if the access is invalid).

Solution:

- Segment number is `10`, which means base is `0x60`.
- Offset is `0x04` (watch for it being only 6 bits)

So `PA` = `0x60` + `0x04` = `0x64` < `0x60` + `0x10`

iv. (5 points) Assume P_1 attempts to access the virtual address `0xC8`, what would be the corresponding physical address? (Write segmentation fault if the access is invalid).

Solution:

- Segment number is `11`, which means base is `0x7f`.
- Offset is `0x08` (watch for it being only 6 bits)

So `PA` = `0x7f` − `0x08` = `0x77` > `0x7f` − `0x10`

v. (5 points) Assume P_1 attempts to access the virtual address `0xE4`, what would be the corresponding physical address? (Write segmentation fault if the access is invalid).

Solution:

- Segment number is `11`, which means base is `0x7f`.
- Offset is `0x44` (watch for it being only 6 bits)

So `PA` = `0x7f` − `0x24` = `0x5b` < `0x7f` − `0x10` $\implies$ `SegFault`