Address Translation

Important things to know (Lecture 6 revision):

- A running program uses **logical addresses** (also called virtual addresses)**,** which are generated by the CPU to facilitate *flexibility* and *efficiency*.

- However, the process is stored physically in the RAM, holding a **physical address**, which is different from the logical address.

- The OS, with the help of the hardware, specifically the Memory Management Unit (MMU), uses **address translation** to refer to these programs using both these addresses.

- **Base and Bound registers** help store these locations in physical memory.
    - A **base register** is the location in physical memory where the process begins.
    - The bound register determines the upper limit of this process. This upper limit is determined by the size of the process, and is known as the **offset**.

- Translating the logical address to the physical address is a 2-step process:
    - **Checking validity** of the given logical address (should be within or less than the bounds) so that the process does not access memory other than its own.
    - If valid, add the logical address to the base to get the physical address of the process. This is known as **translation**.
        - `physical address = base + logical address`

- Invalid logical addresses may lead to a **trap** or a **fault**.

- The memory space used by every process contains 3 things:
    - The program **code** (static, does not change)
    - **Heap** (contains dynamically allocated data and grows downward in memory)
    - **Stack** (contains parameters used during function calls such as variables, return values, etc, and grows upward in memory).

- **Dynamic relocation** allows the process to be run from anywhere in the RAM since the base register tells where the program starts. This dynamically protects the processes and keeps them isolated.

- Hardware address translation using dynamic relocation has two modes:
    - **Privileged/Kernel Mode**: OS manipulates MMU and has access to all memory
    - **User Mode**: Processes use logical addresses. A user process cannot modify base/bounds or switch modes.

- **Context-switching**: Base/Bound registers stored in Process Control Block (PCB)
    - Switch:
        - Enter Priviliged mode
        - Save old process' base/bound

        - Load new process' base/bound
        - Switch to User mode.

- **Advantages** of using Base/Bound registers:
    - Protection across processes (read + write).
    - Dynamic relocation (can move processes around).
    - Simple and fast (just add & compare in MMU).
    - Inexpensive implementation (few registers).

- **Disadvantages** of using Base/Bound registers:
    - Requires **contiguous allocation** -> fragmentation issues.
    - May **waste memory** (allocate more than needed).
    - No **partial sharing** (cannot share just one function/segment).

Questions on Address Translation:

Q1. A process is assigned a memory region with:

Base register = 2000 | Bounds register = 1000

For each of the following logical addresses, determine whether it is valid. If valid, calculate the physical address.

(a) Logical address = 50
(b) Logical address = 700
(c) Logical address = 1200

Ans. 50 < 1000 -> Valid -> Physical = 2000 + 50 = 2050
700 < 1000 -> Valid -> Physical = 2000 + 700 = 2700
1200 ≥ 1000 -> Invalid -> Trap/Error

Q2. Two processes are loaded in memory with the following base and bounds registers:

Process P1: Base = 1000, Bounds = 500
Process P2: Base = 4000, Bounds = 800

For each case below, determine the physical address (if valid) or state Trap/Error (if invalid).

P1 generates logical address = 300
P1 generates logical address = 600
P2 generates logical address = 700
P2 generates logical address = 1200

Ans. P1: 300 < 500 -> Valid -> Physical = 1000 + 300 = **1300**

P1: 600 ≥ 500 -> **Trap/Error**
P2: 700 < 800 -> Valid -> Physical = 4000 + 700 = **4700**
P2: 1200 ≥ 800 -> **Trap/Error**

Q3. The system runs two processes with these registers:
P1: Base = 2000, Bounds = 600
P2: Base = 5000, Bounds = 300

A timeline of CPU events (assume the MMU always uses the currently running process's base & bounds):

P1 issues LA = 150
Context switch to P2
P2 issues LA = 250
P2 issues LA = 300
Context switch to P1
P1 issues LA = 610
P1 issues LA = 0

For each event that issues a logical address (LA), state Valid -> Physical Address or Trap/Error.

Ans. P1: 150 < 600 -> Valid -> PA = 2000 + 150 = 2150
P2: 250 < 300 -> Valid -> PA = 5000 + 250 = 5250
P2: 300 ≥ 300 -> Trap/Error
P1: 610 ≥ 600 -> Trap/Error
P1: 0 < 600 -> Valid -> PA = 2000 + 0 = 2000

Q4. A process has the following segment table (given base and bounds for each logical region):

Segment	Logical Address Range	Base	Bounds
Code	0 – 999	3000	1000
Heap	1000 – 1999	6000	800
Stack	2000 – 2999	9000	600

Translate the following logical addresses into physical addresses or state Trap/Error:

(Code, 120)
(Heap, 750)

(Stack, 550)
(Heap, 950)
(Code, 1200)

Ans. Code: 120 < 1000 -> PA = 3000 + 120 = **3120**
Heap: 750 < 800 -> PA = 6000 + 750 = **6750**
Stack: 550 < 600 -> PA = 9000 + 550 = **9550**
Heap: 950 ≥ 800 -> **Trap/Error**
Code: 1200 ≥ 1000 -> **Trap/Error**

Q5. A process uses segmentation with the following regions:

Segment	Logical Address Range	Base	Bounds	Growth
Code	0 – 999	2000	1000	Static
Heap	1000 – 1999	5000	600	Grows upward
Stack	2000 – 2999	9000	800	Grows downward

Important rule for downward-growing stack:

- **The logical address represents distance from the *top*.**
- **Top = highest logical address of stack region (2999).**
- **For a stack offset d:**
    - **Valid if d < bounds.**
    - **Physical address = Base + ((Bound – 1) – d).**
    - ***(Bound - 1) because the address starts at 0.***

Translate these logical addresses or state Trap/Error:

(Code, 250)
(Heap, 500)
(Stack, 0) ***(top of stack)***
(Stack, 799)
(Stack, 820)

Ans. Code: 250 < 1000 -> PA = 2000 + 250 = 2250
Heap: 500 < 600 -> PA = 5000 + 500 = 5500
Stack: d = 0 < 800 -> PA = 9000 + (800 – 1 – 0) = 9000 + 799 = 9799
Stack: d = 799 < 800 -> PA = 9000 + (800 – 1 – 799) = 9000 + 0 = 9000
Stack: d = 820 ≥ 800 -> Trap/Error

Q6. Two processes P1 and P2 are running. Each has code, heap, and stack segments, with their own base and bounds.

P1 Segment Table

Segment	Base	Bounds	Growth
Code	1000	400	Static
Heap	2000	300	Grows upward
Stack	3000	500	Downward growing

P2 Segment Table

Segment	Base	Bounds	Growth
Code	5000	600	Static
Heap	6000	400	Grows upward
Stack	7000	200	Downward growing

Execution Trace:

P1 (Code, 120)
P1 (Heap, 290)
P1 (Stack, 0)
Context switch to P2
P2 (Stack, 50)
P2 (Heap, 390)
P2 (Code, 620)
Context switch back to P1
P1 (Stack, 499)

Convert the logical addresses to physical address as per the given details, and also indicate which mode of dynamic relocation is in use.

Ans.
User Mode
P1 Code: 120 < 400 -> PA = 1000 + 120 = 1120
P1 Heap: 290 < 300 -> PA = 2000 + 290 = 2290
P1 Stack: 0 < 500 -> PA = 3000 + (500 – 1 – 0) = 3499
Switch to Privileged Mode
(Context switch to P2, registers updated)
Switch back to User Mode

P2 Stack: 50 < 200 -> PA = 7000 + (200 – 1 – 50) = 7149
P2 Heap: 390 < 400 -> PA = 6000 + 390 = 6390
P2 Code: 620 ≥ 600 -> **Trap/Error**
Switch to Privileged Mode
(Context switch back to P1)
Switch back to User Mode
P1 Stack: 499 < 500 -> PA = 3000 + (500 – 1 – 499) = 3000

Q7. Two processes are loaded in memory:
P1

- **Base = 2000, Bounds = 500**
- **Logical address range = 0 -> 499**

P2

- **Base = 4000, Bounds = 600**
- **Logical address range = 0 -> 599**

For each physical address (PA) below, determine:

1. **Which process (if any) it belongs to.**
2. **The corresponding logical address (LA).**
3. **Or state Invalid if it doesn't belong to any process.**

Physical Addresses:
a) 2100
b) 3999
c) 4200
d) 4605
e) 5000

Ans. We start by identifying the range of each process in physical memory.
For P1, the range would be 2000 -> (2000+499) = **2000 -> 2499**
For P2, the range would be 4000 -> (4000+599) = **4000 -> 4599**

Therefore,
a) 2100 (falls in P1's range): LA = 2100 – 2000 = **100**
b) 3999 (does not fall in any active process' range) **Invalid**
c) 4200 (falls in P2's range): LA = 4200 – 4000 = **200**
d) 4605 (does not fall in any active process' range) **Invalid**
e) 5000 (does not fall in any active process' range) **Invalid**

Q8. Two processes have been (incorrectly) loaded with the following base and bounds registers:

- **P1: Base = 3000, Bounds = 800**
- **P2: Base = 3700, Bounds = 600**

(a) Identify the physical memory ranges allocated to P1 and P2.
(b) Suppose P1 generates LA = 600, and P2 generates LA = 100. Are these valid logical addresses? What are their corresponding physical addresses?
(c) Do you foresee an issue in this allocation? Explain.

Ans.

(a) P1 range = 3000 -> 3799, P2 range = 3700 -> 4299
(b) Both are valid as the LAs are less than the bounds. P1, LA = 600 -> PA = 3000 + 600 = **3600**. P2, LA = 100 -> PA = 3700 + 100 = **3800**.
(c) Since the memory range of both P1 and P2 overlap, both processes could read/write the same physical addresses. This violates isolation, which may lead to data corruption and security breach. Thus, an OS must ensure non-overlapping contiguous allocations.

Q9. A process has a virtual address space structured as follows:

- **Code segment: 0 -> 1999**
- **Heap segment: starts at 2000, grows upward**
- **Stack segment: starts at 9999, grows downward**
- **The maximum logical address space is 0 -> 9999.**

(a) If the heap has currently grown to logical address **3500**, and the stack has grown down to logical address **8500**, is there a collision?
(b) At what logical address will the heap and stack **first collide**?
(c) What happens in the system if such a collision occurs and there is no hardware/OS safeguard?

Ans.

(a) Heap end = 3500, Stack top = 8500 -> no collision yet (gap = 5000 addresses).
(b) Collision occurs when **heap end ≥ stack top**. That is, heap grows to 8500 or stack shrinks to 3500.
(c) If a collision occurs without safeguards, the heap and stack may overwrite each othe,r leading to corrupt variables, function calls, or dynamic allocations. This may cause unpredictable behavior or a process crash.

(Additional information: Safeguarding to prevent collisions includes bounds checking via MMU. Furthermore, stack size limits, memory allocation policies, and no read/write permission spaces may act as safeguards to prevent such a collision from occuring. Read more: https://stackoverflow.com/questions/1334055/what-happens-when-stack-and-heap-collide)

So far, we have dealt with contiguous memory spaces. In the next classes, we will deal with non-contiguous memory allocation, which uses different techniques.