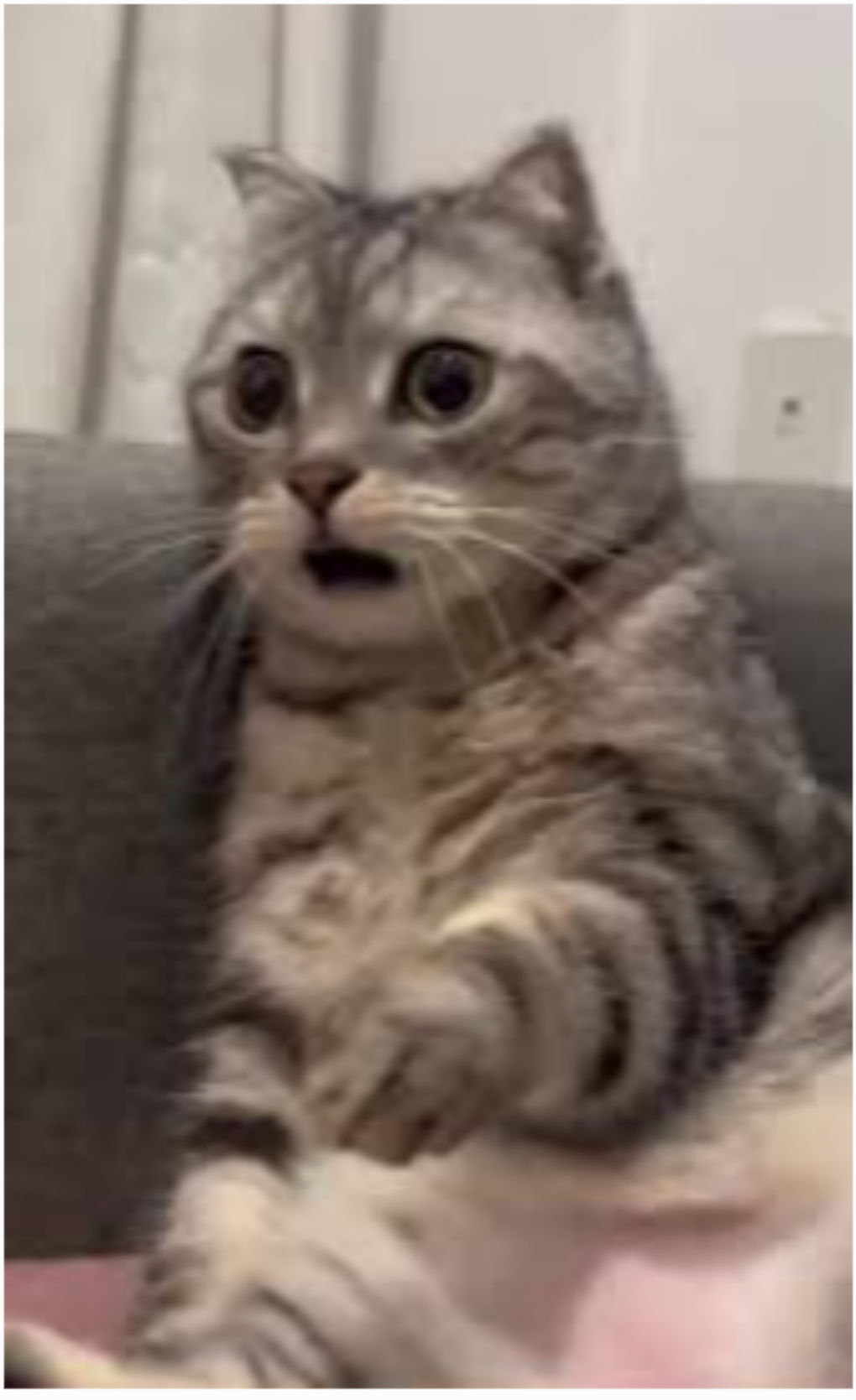
11. Address Translation



OS ensures that no application is allowed to access any memory but its own

=> How to Efficiently & Flexibly Virtualize Memory ?

: Hardware-based address translation

Hardware-based address translation

the hardware transforms each memory access
(ex. an instruction fetch, load, or store)

virtual address (provided by the instruction) -> physical address (where the desired information is actually located)

the **OS** must get involved at key points to set up the hardware
=> ensure correct translations take place => manage memory

- keeping track of which locations are free & which are in use
- judiciously intervening to maintain control over how memory is used

many programs are actually sharig memory at the same time

Dynamic (Hardware-based) Relocation

2 hardware registers within each CPU
: base register / bounds(limit) register

when a program starts running,
1. the OS decides where in physical memory it should be loaded
2. the OS sets the base register to that value.

physical address = virtual address + base

Each memory reference generated by the process is a **virtual address** **physical address** that can be issued to the memory system

- base register : transform virtual addresses -> physical addresses
- bounds (limits) register : ensures addresses are within the confines of the address space

Why dynamic relocation?

- ths relocation of the address happens at runtime
- we can move address spaces even after the process has started running

the base & bounds registers are hardware structures kept on the chip (**one pair per CPU**), part of MMU

MMU (Memory Management Unit) : part of processor that helps with address translation

Two type of bound register

1. holds the size of the address space
   => the hardware checks the virtual address before adding the base
2. holds the physical address of the end of the address space
   => the hardware first adds the base -> makes sure the address is within bounds

Q. Why is second one even used?
it seems obvious that former one is way much eaiser since it doesn't need to add up base address.
: **Physical address space constraints**

Some systems can only allocate a process to a specific range of physical addresses.

For example, if base = 1000 and bound = 2000, then the valid physical address range is 1000 ~ 2000.

In this case, when the hardware adds the base to the virtual address, it must ensure that the resulting **physical address** stays within this range.

Therefore, checking the **physical address** against the bound register can be more straightforward and practical.

Software-Based (Static) Relocation
the loader would rewrite the instruction to offset each address by base value

- It doesn't provide protection, as processes can generate bad addresses and thus illegally access other process's or even OS memory
- once placed, it is difficult to later relocate an address space to another location

Modify the base & bounds registers

the hardware should provide special instructions to modify the base & bounds registers, allowing the OS to change them when different processes run.

These instructions are privileged
Only in kernel ode can the registers be modified.
Imagine the havoc a user process could wreak if it could arbitrarily change the base register.

The free list

The OS must track which parts of free memory are not in use, so as to be able to allocate memory to processes
the simplest is a free list, which simply is a list of the ranges of the physical memory which are not currently in use

Generate exceptions

CPU must be able to generate exceptions in situations where a user program tries to access memory illegally (with an address that is "out of bounds")
In this case, CPU should stop executing the user program and arrange for the OS "out-of-bounds" exception handler to run.

Similarly, if a user program tries to change the values of the (privileged) base and bounds registers, the CPU should raise an exception and run the OS handler.

Hardware Requirements	Notes
Privileged mode	*Needed to prevent user-mode processes from executing privileged operations*
Base/bounds registers	*Need pair of registers per CPU to support address translation and bounds checks*
Ability to translate virtual addresses and check if within bounds	*Circuitry to do translations and check limits; in this case, quite simple*
Privileged instruction(s) to update base/bounds	*OS must be able to set these values before letting a user program run*
Privileged instruction(s) to register exception handlers	*OS must be able to tell hardware what code to run if exception occurs*
Ability to raise exceptions	*When processes try to access privileged instructions or out-of-bounds memory*

Figure 15.3: **Dynamic Relocation: Hardware Requirements**

Operating System Issues

Free space allocation : when a process is created

the OS must take action when a process is created, finding space for its address space in memory.

Given our assuptions that each address space is

1. smaller than the size of physical memory
2. the same size

this is quite easy for the OS.

It can simply view physical memory as an array of slots, and track whether each one is free or in use.

When a new process is created, the OS will have to search a data structure (often called a free list) to find room for the new address space and then mark it used.

With variable-sized address spaces, life is more complicated, but we will leave that concern for future chapters.

Reclaim memory : when a process is terminated

OS must do some work when a process is terminated, reclaiming all of its memory for use in other processes or the OS.

The OS thus puts its memory back on the free list, and cleans up any associated data structures as need be.

OS Requirements	Notes
Memory management	*Need to allocate memory for new processes;* *Reclaim memory from terminated processes;* *Generally manage memory via* ***free list***
Base/bounds management	*Must set base/bounds properly upon context switch*
Exception handling	*Code to run when exceptions arise;* *likely action is to terminate offending process*

Figure 15.4: **Dynamic Relocation: Operating System Responsibilities**

When a context switch occurs

There is only one base & bounds register pair on each CPU.
Their values differ for each running program, as each program is loaded at a different physical address in memory.

Thus the OS must save and restore the base-and-bounds pair when it switches between processes.

Specifically, when the OS decides to stop running process, it must save the values of the base and bounds registers to memory, in some per-process structure such as the process structure or process control block (PCB).

Similarly, when the OS resumes a running process (or runs it the first time), it must set the values of the base and bounds on the CPU to the correct values for this process.

We should note that when a process is stopped, it is possible for the OS to move an address space from one location in memory to another rather easily.

To move a process's address space,

1. the OS first deschedules the process.
2. Then, the OS copies the address space from the current location to the new location.
3. Finally, the OS updates the saved base register (in the process structure) to point to the new location.

When the process is resumed, its (new) base register is restored, and it begins running again, oblivious that its instructions and data are now in a completely new spot in memory.

the OS installs exception handlers at boot time (via privileged instructions)

OS @ boot (kernel mode)	Hardware	(No Program Yet)
initialize trap table		
	remember addresses of...	
	system call handler	
	timer handler	
	illegal mem-access handler	
	illegal instruction handler	
start interrupt timer		
	start timer; interrupt after X ms	
initialize process table		
initialize free list		

Figure 15.5: **Limited Direct Execution (Dynamic Relocation) @ Boot**

note how its memory translations are handled by the hardware with no OS intervention.

In most cases, the OS just sets up the hardware appropriately and lets the process run directly on the CPU. Only when the process misbehaves does the OS have to become involved.

Summary

limited direct execution - extended -> address translation

With address translation, the OS can control each and every memory access from a process, ensuring the accesses stay within the bounds of the addres space.

Key to the efficiency : hardware support, translation quickly for each access, turning virtual addresses -> physical ones.

All of this performed in a way that is transparent to the process that has been relocated.

Base-and-bounds virtualization : efficient
only a little more hardware logic is required to add a base register to the virtual address and check that the address generated by the process is in bounds.
& also offers protection

OS @ run (kernel mode)	Hardware	Program (user mode)
To start process A:		
allocate entry in process table		
alloc memory for process		
set base/bound registers		
return-from-trap (into A)		
	restore registers of A	
	move to **user mode**	
	jump to A's (initial) PC	
		Process A runs
		Fetch instruction
	translate virtual address	
	perform fetch	
		Execute instruction
	if explicit load/store: ensure address is legal	
	translate virtual address	
	perform load/store	
		(A runs...)
	Timer interrupt	
	move to **kernel mode**	
	jump to handler	
Handle timer		
decide: stop A, run B		
call `switch()` routine		
save regs(A) to proc-struct(A) (including base/bounds)		
restore regs(B) from proc-struct(B) (including base/bounds)		
return-from-trap (into B)		
	restore registers of B	
	move to **user mode**	
	jump to B's PC	
		Process B runs
		Execute bad load
	Load is out-of-bounds;	
	move to **kernel mode**	
	jump to trap handler	
Handle the trap		
decide to kill process B		
deallocate B's memory		
free B's entry in process table		

Figure 15.6: **Limited Direct Execution (Dynamic Relocation) @ Runtime**