MIDTERM REVIEW SESSION

Kai Mast

CS 537

Fall 2022

QUESTION 1: BASE AND BOUNDS

Professor McFlub: "Base/bounds-based virtual memory is really easy. Imagine you have a base register and a bounds register in each CPU. The base points to the physical memory location where an address space is relocated; the bounds tells us how big such an address space can be. Let's do an example to understand this better.

Assume we have the following base/bounds pair:

Base: `0x1000` Bounds: `0x10`

Now assume we have the following virtual memory references by a process: `0x0, 0x4, 0x8, 0xc`

The corresponding physical addresses that will be referenced are:

`0x1000, 0x1004, 0x1008, FAULT` (because this one is out of bounds)

Make sense?"

No! `0xc < 0x10` (Not out of bounds)

0xc should have translated to 0x100c

Question 5 from Fall '13 Midterm 1

QUESTION 2: PAGING, PART 1

Assume the following: a 32-bit virtual address space, with a 1KB page size.

Basics:

How many bits are in the offset portion of the virtual address?

10 bits (for a 1KB page, you need 10 bits to address each byte)

How many bits are in the VPN portion of the virtual address?

That leaves 22 bits for the VPN (32 bits total - 10 bits of offset)

Question 3 from Spring 09 Midterm

QUESTION 2: PAGING, PART 2

Assume the following: a 32-bit virtual address space, with a 1KB page size. Assume each page table entry is 4 bytes in size. Assuming a **linear page table**.

How many entries are in the table? 2^{22} *(one for each virtual page)*

What is the total size of the table? $2^{22} \times 4bytes = 16MB$

In a live system, how much memory would be occupied by page tables? (what factors affect this?)

*16MB **per process**. If there are 100 processes, for example, this implies 1600MB of page table space is needed!*

QUESTION 2: PAGING, PART 3

Assume the following: a 32-bit virtual address space, with a 1KB page size. Assume each page table entry is 4 bytes in size. Assuming a **page table with two levels.**

How many PTEs fit onto a single page in this system? *1KB/4B = 256*
How many bits (in the VPN) are needed to index the page tables?
8 (because 2^8 = 256)

How many bits (in the VPN) are needed to index the page directory?
14 (VPN is 22bits, so 22-8=14)

How much memory is needed for the page directory? (each PDE is 4bytes)
64KB (2^14 total entries, each is 4 bytes)

QUESTION 2: PAGING, PART 4

Given the following memory allocation, write down both (a) how much memory our multi-level page table consumes and (b) how much memory a linear page table consume.

Code is located at address 0 and there are 100 4-byte instructions. The heap starts at page 1 and uses 3 total pages. The stack starts at the other end of the address space, grows backward, and uses 3 total pages.

Multi-level Page Table Size?

- Each chunk covers 256 consecutive pages; so code pages and heap in the same chunk
- The stack, at the other end of the address, so it needs its own chunk
- Hence 2 pages (1KB each) plus the page directory (64KB), or 66KB

Linear Page Table Size? 16MB (always stays the same)

QUESTION 2: PAGING, PART 5

Given the following memory allocation, write down both (a) how much memory our multi-level page table consumes and (b) how much memory a linear page table consume.

Code is located at address 0 and there are 100 4-byte instructions. The heap starts at page 1 and uses 1000 total pages. The stack starts at the other end, grows backwards, and uses 1000 total pages.

Multi-level Page Table Size?

- We need four pages/chunks of the page table to translate the first 1001 pages of the address space (four pages of the page table covers 1024 pages) and another four pages to translate the last 1000
- Hence, 8 pages (1KB each) plus the page directory (64KB) gets us to 72KB

Linear Page Table Size? 16MB (always stays the same)

QUESTION 3, MLFQ

The Multi-level Feedback Queue (MLFQ) is a fancy scheduler that does lots of things. Which of the following things is true about the MLFQ approach?

MLFQ learns things about running jobs
True: By moving jobs down queues, it learns that they are long running (for example).

MLFQ starves long running jobs **False:** By bumping priority on occasion MLFQ avoids starvation.

MLFQ uses different length time slices for job **True:** sometimes, across different queues.

MLFQ uses round robin **True:** within a given level

MLFQ forgets what it has learned about running jobs sometimes
True: With priority bump to top priority, all jobs look the same now, and all is forgotten about them.

Problem 11 from Spring '18 Midterm

QUESTION 4: "IRON TLB"

Sometimes a bit gets flipped and thus the wrong translation ends up in the TLB! For each of the following fields, both (1) describe what the field is used for and (2) explain what you think would happen if a bit gets flipped in said field just before the OS installs the entry in the TLB.

Problem 4 from Spring '10 Midterm

QUESTION 4: "IRON TLB"

What is the VPN field for?

- TLB entries are not in order
- Needed to indicate the virtual page this TLB mapping is for

What would happen if a bit in the VPN got flipped?

- Virtual address might get mapped to the wrong physical page
- We might get a TLB miss

QUESTION 4: "IRON TLB"

What is the ASID field for?

- Indicates the process this entry is for
- Prevents us from flushing the entire TLB after a context switch

What would happen if a bit in the ASID got flipped?

- TLB might use the an entry meant for another process
- Or we might get a TLB miss

QUESTION 4: "IRON TLB"

What is the Valid bit for?

- Indicates if the TLB entry is valid or not
- There can still be a valid page table entry, even if the TLB entry is invalid!

What would happen if the valid bit got flipped?

- We might get a TLB miss (if flipped to false)
- We might resolve to an invalid physical page (if flipped to true)

QUESTION 5: (MORE) TLB

TLBs are a critical part of modern paging systems. Assume the following system:

- Page size is 64 bytes
- TLB contains 4 entries
- TLB replacement policy is LRU (least recently used)

Each of the following represents a virtual memory address trace,
i.e., a set of virtual memory addresses referenced by a program. In which of the following traces will the TLB possibly help speed up execution?

Problem 17 from Spring ‘18 Midterm

QUESTION 5: TLB

TLBs are a critical part of modern paging systems. Assume the following system:
Page size is 64 bytes, TLB contains 4 entries, TLB replacement policy is LRU

0, 100, 200, 1, 101, 201, ... (repeats in this pattern)
Speed up: Three pages accessed repeatedly; first miss, miss, miss, but then hit, hit hit, etc.
0, 100, 200, 300, 0, 100, 200, 300, ... (repeats)
Speed up: 4 pages being accessed repeatedly, misses the first time, then hits.
0, 1000, 2000, 3000, 4000, 0, 1000, 2000, 3000, 4000, ... (repeats)
No Speedup: 5 pages being accessed in a cyclical pattern; each access is a TLB miss.
0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, ... (repeats)
Speed up: Just one page being accessed repeatedly, so mostly hits (except first access).
300, 200, 100, 0, 300, 200, 100, 0, ... (repeats)
Speed up: 4 pages accessed repeatedly, hence mostly hits.

QUESTION 6: TRAPS AND INTERRUPTS

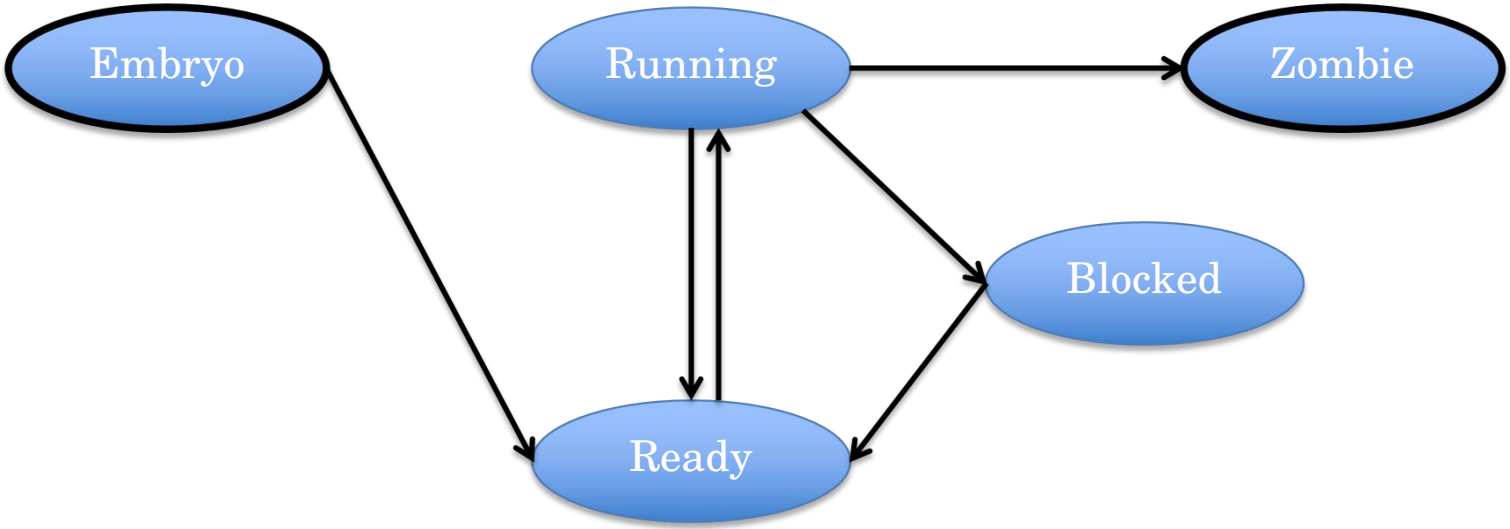
Professor McFlub: *"Traps, like system calls, and interrupts, such as from a programmable timer or disk, are handled very similarly in OSes. For example, when each occurs, the hardware saves some state (such as the program counter) and jumps into the OS. At that point, the OS handles the trap or interrupt. When finished, the OS returns, through a normal return instruction, and then retries the trapping or interrupted instruction."*

Whats wrong with or incomplete about this statement?

- Returns from the kernel require a different instruction, which both undoes what the trap did (retores saved state), and changes privileged mode back to user mode
- Retry only happens in some cases (e.g., TLB miss); in other cases (e.g., syscall trap), the return must return to the instruction after the trapping instruction

Question 4 from Fall '13 Midterm

QUESTION 7: PROCESS STATES EXTENDED

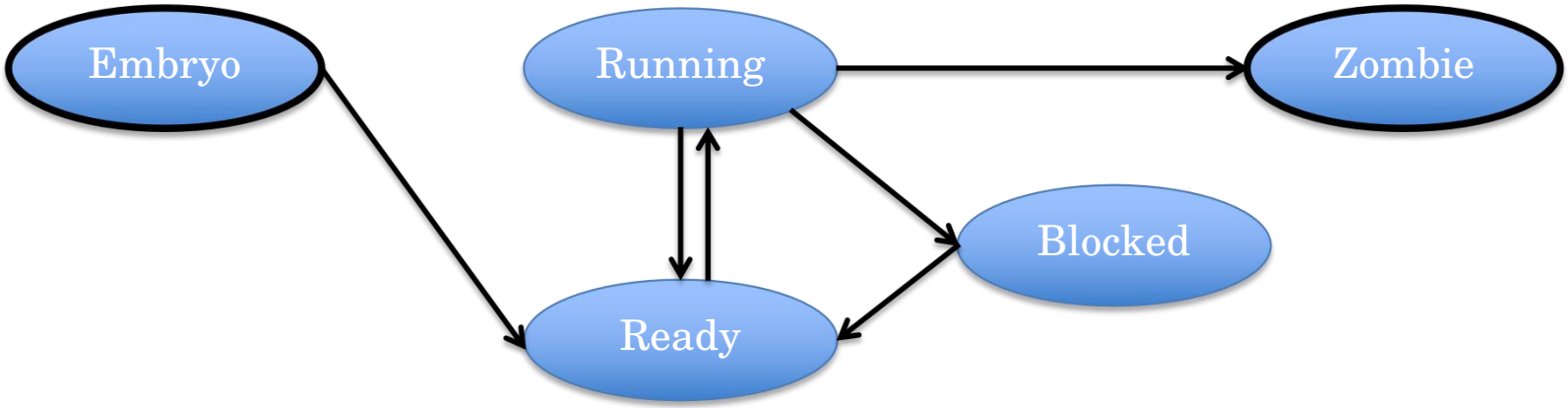


Assuming you start observing the states of a given process at some point in time (not necessarily from its creation, but perhaps including that), which process states could you possibly observe?

Note: once you start observing the process, you will see ALL states it is in, until you stop sampling.

Problem 4 from Spring '18 Midterm

QUESTION 7: PROCESS STATES EXTENDED



Which sequence of states is possible?

Running, Running, Running, Ready, Running, Running, Running, Ready	Possible
Embryo, Ready, Ready, Ready, Ready, Ready	Possible
Running, Running, Blocked, Blocked, Blocked, Running	Not Possible
Running, Running, Blocked, Blocked, Blocked, Ready, Running	Possible
Embryo, Running, Blocked, Running, Zombie, Running	Not Possible