CS2106
Introduction to OS

Office Hours, Week 9

(includes some week 8 questions)

Agenda

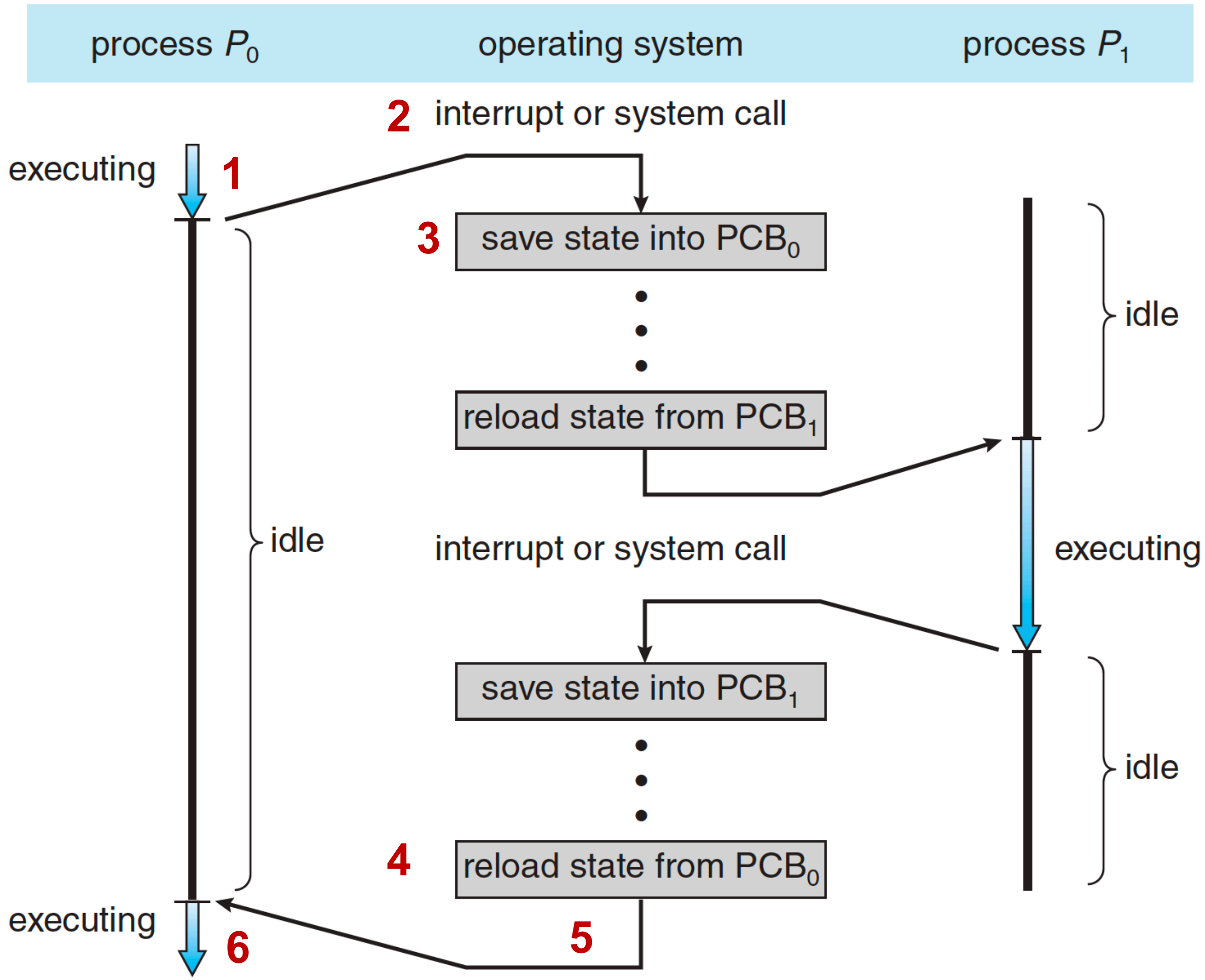
- Mock Midterm Quiz 2
- Week 7 questions on Memory Abstraction
- Week 8 questions on Contiguous Allocation
- Week 8 questions on Disjoint Allocation

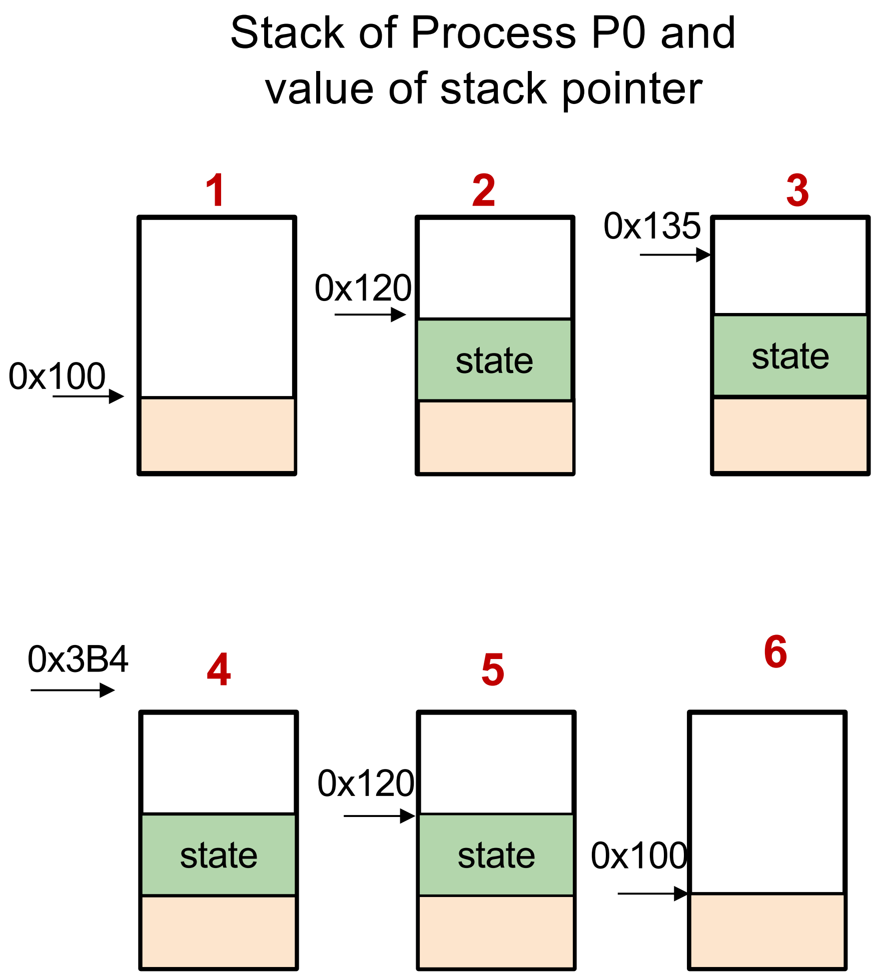
- Note: Week 7 synchronization questions excluded
  - there will be a separate revision session to cover synchronization questions + some exercises

Question from **Mock Exam Quiz 2**

Process P has been running on System X, which does Round-Robin scheduling. After P's time quantum expired, P transitions from state RUNNING to state READY. Immediately after the transition, the process control block (PCB) of process P will contain: :

- A) The content of P’s stack as it was immediately before the transition.
- B) The content of P’s stack as it was immediately after the transition.
- C) The value of P’s stack pointer as it was before the transition.
- **D) The value of P’s stack pointer as it was after the transition.**
- E) None of the other answers is correct.





Check the notes for explanation

Memory Abstraction

What does the "*loading time*" of a process means? Is it similar to the runtime?

- The time it takes the OS to load the process from the disk (where it’s stored as a file such as .exe, a.out, etc.), and set up the necessary OS structures.
- This happens before the process starts the execution.
- It may or may not be included in total runtime, depends on how you measure it.

Memory Abstraction

How is the performance of Logical Address as compared to Base + Limit Registers?

- Logical addressing isn't a specific technique
- Many techniques, including the *Base+Limit* registers, actually use the separation of logical and physical addresses

Memory Abstraction

How is using base register different from adding offset to memory references?

- Without base/limit registers, the OS has to fix every program before loading by adding offsets, which may take time
- Using the base/limit register eliminates the need for OS inspection of the code and speeds up loading

Memory Abstraction

When you say the assumption is partition fits in memory, does the memory refers to RAM / primary storage?

- Refers to RAM (main memory)
- Usually made in DRAM technology

When discussing memory, are we talking only about RAM or can it include others (e.g., HDD)?

- When we say just **memory**, it’s not clear
- When we talk about **physical memory**, then it’s RAM
- But when we talk about **virtual memory** (next lecture), it can mean storage too.

Dynamic Partitioning (Continuous Allocation)

In dynamic partitioning, when we search for a hole of size > N, why not >=N?

- Good catch!
- We should search for a hole >=N

Dynamic Partitioning (Continuous Allocation)

If the memory size of a process is unknown, how is the allocation of partitions done? Is there a way to dynamically allocate new partitions, which will be covered later?

- Good question!
- If process needs more dynamically allocated memory than provisioned, more memory needs to be given to the process
- In case of contiguous allocation, the partition may be extended
- Otherwise needs to be relocated into a larger partition

Dynamic Partitioning (Continuous Allocation)

For the dynamic partitioning on slide 31, instead of just deleting A, why not delete both A and B, and then add B to create a single big hole?

- What you are suggesting is basically compaction
- We can do that, but it's very computationally intensive
  - And has limitations (e.g., it's tricky to move data that has pointers to it)
- It's usually done when the system cannot allocate a partition for a new process

Buddy Allocation

If the process needs 100MB, and the partition size is 128MB, can the remaining 28MB be allocated to something else?

- Not in the buddy system
- This becomes internal fragmentation
  - Allocated to the process, but unused

Buddy Allocation

Does each slot in the array A[0..K] in buddy system stores a linked list of addresses of *free* blocks? Or does each list also contain the occupied memory regions and uses a flag to determine which is occupied/free?

- Great question
- It’s enough to keep only the free blocks
  - although there are many implementations, which which may differ on this issue, or whether or not the lists are sorted
- A separate bitmap (one per each level) can be kept to indicate whether a partition is free or occupied
  - Helps to check quickly if your buddy is free

Buddy Allocation

Why is the deallocation complexity in Buddy system O(N)?

- Imagine there are N partitions in the smallest bucket
- It may take also O(N) time to find out if your buddy is free
  - Keeping the list sorted by the address may help
  - Recall that you know the address of your buddy, but you don't know its position in the list.
- You need insert your partition into the free list
  - If the list is sorted, it may take O(N) time to find your place
- Can be improved with additional structures (e.g., bitmaps)

Buddy Allocation

For the buddy system, why don't we create our buckets containing memory blocks of size 2^0, 2^1, 2^2 etc. from the start instead of splitting the big blocks only when we need it?

- We can!
- But usually we want to keep free partitions as big as possible
- You can also use something called Slab allocation, which can be layered on top of buddy system
  - Allows for fast and efficient allocation of objects of commonly used sizes
  - (Non-examinable)

Buddy Allocation

Lecture 7 slide 37: How do you get the size of A, B and C?

Lecture Context:

Block address **A** is in the format $\texttt{xxxx00..00}_2$

Splitting **A** into 2 blocks of half the size:

B = $\texttt{xxxx00..00}_2$ and **C** = $\texttt{xxxx1..00}_2$

Answer:

- The partition size is determined based on which list it belongs in
- Given only the starting address of a partition, you cannot know its size
- Example: starting address 0
  - It could be the biggest partition size
  - It could be the smallest partition size
  - It could be any (power of two) partition size!

Buddy Allocation

Lecture 7 slide 39: Why isn't block Q allocated at 256 and not at 0? And what is size = 128?

- Because Q's requested size is 250, it needs a partition of 256
  - Size refers to the allocated partition size
  - Should be 256, not 128! Good catch
- 0-128 is free, but we need a 256 partition
  - That's why we put take one that starts at 256 and ends at 512.

Buddy Allocation

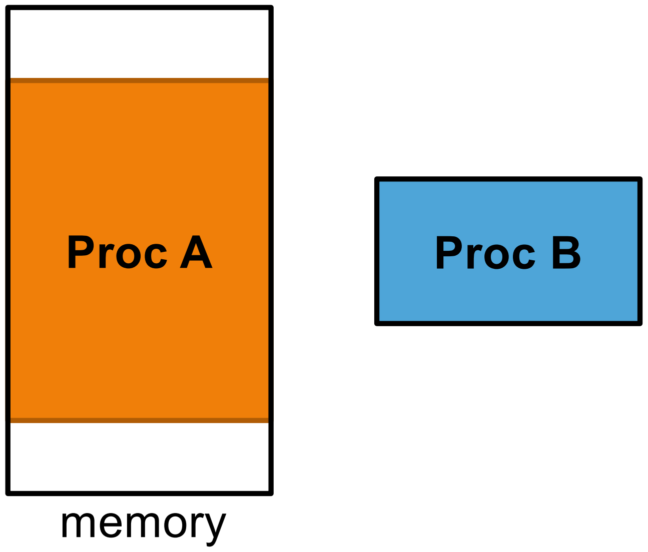
For thew buddy system from mem allocation, I get why we find 2^n that's smaller than S. but if S is sufficiently small, wouldn't it be easier to chose the base to be S instead of 2? can we make the algorithm choose a var x so x^n instead of 2^n?

- Not sure I understand the question
  - Try asking again
- However, we are not limited to use binary partitions
  - binary partitions are the most convenient
  - Also, using other base (3+) means it’s not a buddy system anymore
    - You have multiple buddies and all of them would need to be free in order to merge

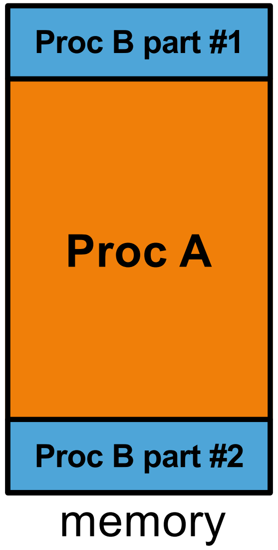
Disjoint Allocation

What is the use of having disjoint chunks in memory?

- Allowing processes to fit that otherwise wouldn't!
- Requiring that a process must fit as one piece is very restricting.



Continuous: enough memory for proc B, but can't fit as a piece



Disjoint allocation can fit both processes

Logical vs. Physical Memory

Is it right to say logical memory exists only within the process and is mapped to real physical memory by the OS?

- Correct!
- The logical memory space (the set of all logical addresses generated by the process) is mapped to physical memory by the OS
- The translation is then done with the help of hardware.

Logical vs. Physical Memory

What are the 2 accesses per memory reference? Is it the logical address and physical address?

- The first access is to the page table, to get the physical address based on logical
- Then using the physical address, we access data

Logical vs. Physical Addresses

Can you repeat in what case do we need more than 2 memory access for `load r1, [2106]`?

- It's a big instruction that includes a full address.
  - Usually doesn't fit in one unit of memory access (architecture dependent)
- One access fetch the first part of the instruction, which says:
  - It's a load instruction
  - The load destination should be register r1
  - An address is needed
- One access to fetch the address 2106; instruction fetching done.
- Third access to fetch the data
- Everything x2 if paging is used

Logical vs. Physical Addresses

Does the PC contain logical address of instructions?

- Yes!
- All addresses generated by the today's processors are logical addresses.

Logical vs. Physical Addresses

Logical address of data is inside the instruction, but where is the logical address of the instructions kept? How does the PC know what the logical address of instructions are?

- Remember that PC holds the logical address of the currently executing instructions
- It starts from the address of the main() function and then follows the control flow
  - Jump to other regions of code (e.g., if/then/else, function calls, loops…)
  - Or continue sequentially

Paging and External Fragmentation

Why there isn't external fragmentation in paging?

- Because a free page can be used regardless of where it is.
- If there are enough free pages to fit a process, the pages could be scattered all over.

Page Table Location

Where in memory is the page table located in?

- Somewhere in the operating system memory, not accessible to user programs.
- More info next week.

Virtual Memory

How does the logical memory space exceed the physical memory space?

- We will see this week in lectures.
- Basically, the logical memory that doesn’t fit into DRAM goes into disk
- It’s called virtual memory

Virtual Memory

Does OS still uses virtual memory since memory is so cheap and abundant nowadays.

- Yes!
- All systems nowadays use virtual memory
- Not so much for the lack of physical memory anymore
- For security and isolation (we will talk about it in the next lecture).

Thank you!