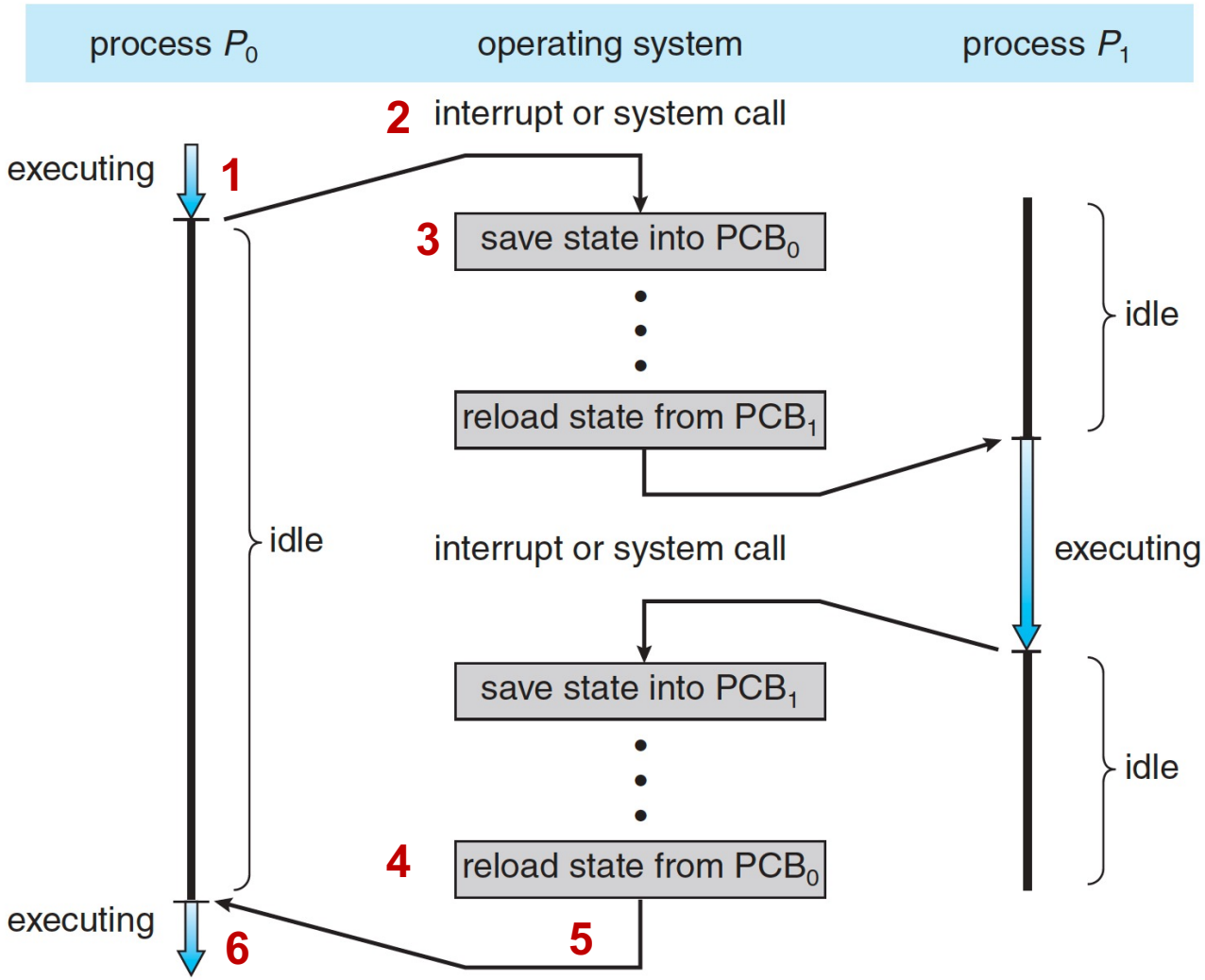
CS2106
Introduction to OS

Office Hours, Week 8 (part 1)

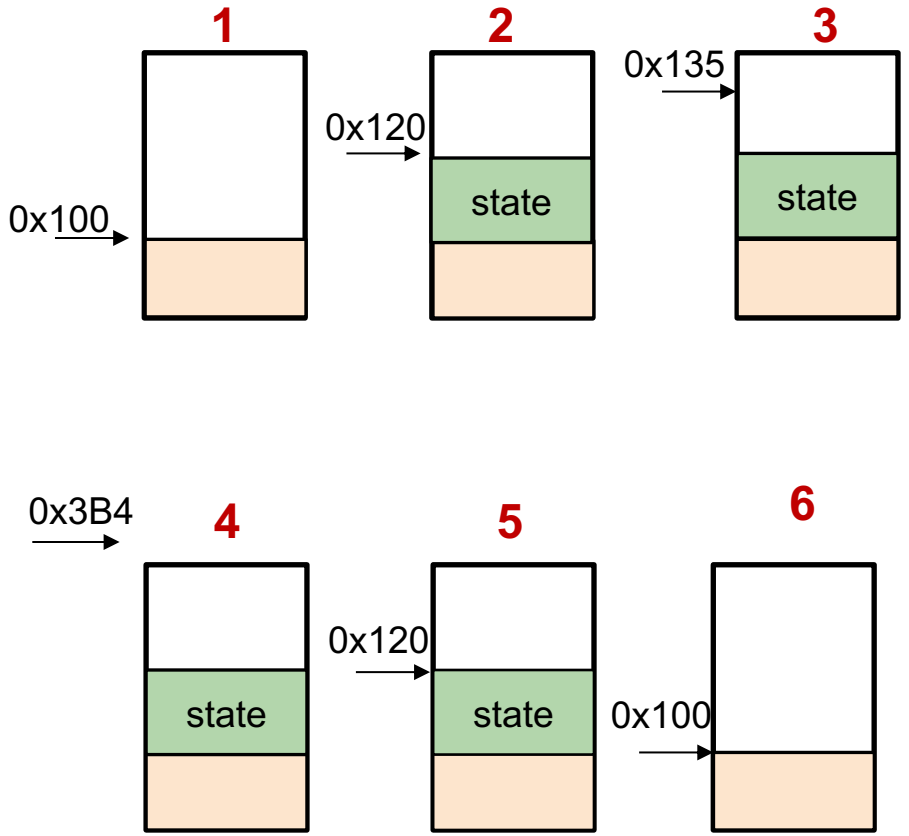
Question from **Mock Exam Quiz 2**

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.



Stack of Process P0 and value of stack pointer



Check the notes for explanation

Midterm

Are the weekly quizzes a good gauge of what to expect for the midterms?

- I think so!
- A bit more clear and less ambiguous, hopefully

Semaphores

Can I access the value inside a semaphore directly? Am I allowed to write pseudocode like: if (semaphore == N) ... ?

- Depends on the implementation
- Different semaphore implementations may or may not provide that functionality
- E.g., in POSIX semaphores, there's a function like:

```
int sem_getvalue(sem_t *sem, int *sval);
```

Semaphores

Why do we need to destroy semaphores, sem_destroy(), after usage and why is freeing memory, free(), not enough?

- A Semaphores is not just an object in memory!
  - It's different from objects you create with malloc() and free()
  - It has some representation/state in the operating system
  - That's why it is good to notify the operating system of its deletion
    - To clear up whatever state there may be in the OS

Semaphores

Is the scheduler involved when deciding the order of the semaphore queue?

- Sometimes yes!
- In some implementations (especially real-time systems), the *signal* semantics can include transferring control directly to the highest priority task in the waiting list
- Excellent question!

Semaphores across processes (not examinable)

So far you have given a lot of examples of semaphore usage to synchronize across threads. Is there a similar mechanism in Unix for synchronization across processes?

What are named and un-named semaphores?

- Two types of semaphores in Unix: *unnamed* and *named*.
  - Unnamed: the usual semaphores, private to a process unless explicitly shared
  - Named: inter-processs shared semaphores identified by *name*.
    - Processes can get access to the semaphore if they know its name
    - Good for synchronization across processes

Alternative to Shared Memory

When using shared data, why not just make copies of the original items and send those copies with a tag about who changed it? Won't this eliminate race condition?

- This basically come down to message passing!
- Convenient for avoiding race conditions, but hard to use
  - Easier to modify variables directly

Peterson's Algorithm

In Peterson's, what would happen if writing to "turn" was NOT an atomic operation? Would the solution fail and why?

- Non-atomic writes can happen when you are writing a big object, so each part has to be written separately. That's not the case here.
- Writing an integer into memory is always atomic
- It is trivially achieved, especially when you want to write 0 and 1.
  - Imagine that two threads want to set turn to 0 and 1, respectively
  - Regardless of what happens, the outcome can be only 0 or 1
    - Cannot be 0.5 or 3
- However, if writing to turn wasn't atomic and somehow suffered from race conditions, then the outcome of writing turn could be 3 or something like that, in which case the solution wouldn't work.

Interactive Scheduling (from Week5)

Regarding MLFQ priority setting rules, when setting the highest priority for new jobs, does new jobs refer to jobs that have entered the ready queue for the very first time, or jobs that are in front of the queue (the next job in the queue to execute)?

- It refers to:
  - Jobs that have just entered the system (brand new)
  - ~~Jobs that come back from I/O and enter the queue~~
- Bringing up as it caused confusion.
- MLFQ has many variants and optimizations, including increasing the priority of a process after I/O
  - This allows processes to occasionally exceed the quantum without being permanently penalized for it
- However, you can ignore all those

Thank you!