CS2106 Assignment 4

Bernard Teo Zhi Yi (bernardteo@u.nus.edu)

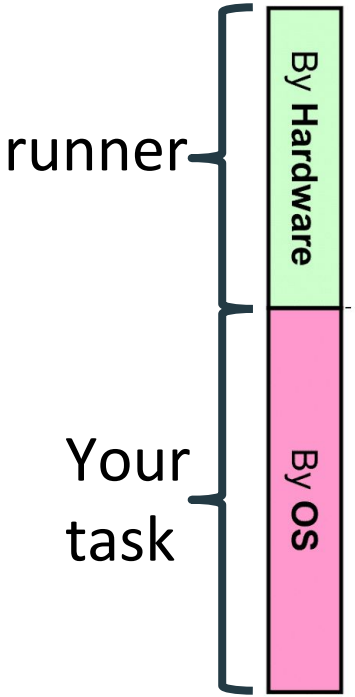
Overview (See LumiNUS announcement)

- There was a bug in runner.c
  - In exec_munmap_and_check_result(), there is a for-loop from 0 to (1<<PAGE_BITS), but it should be from 0 to (1<<FRAME_BITS) instead
- Some additional lab slots have been scheduled next week to make up for the public holiday on Monday

Overview

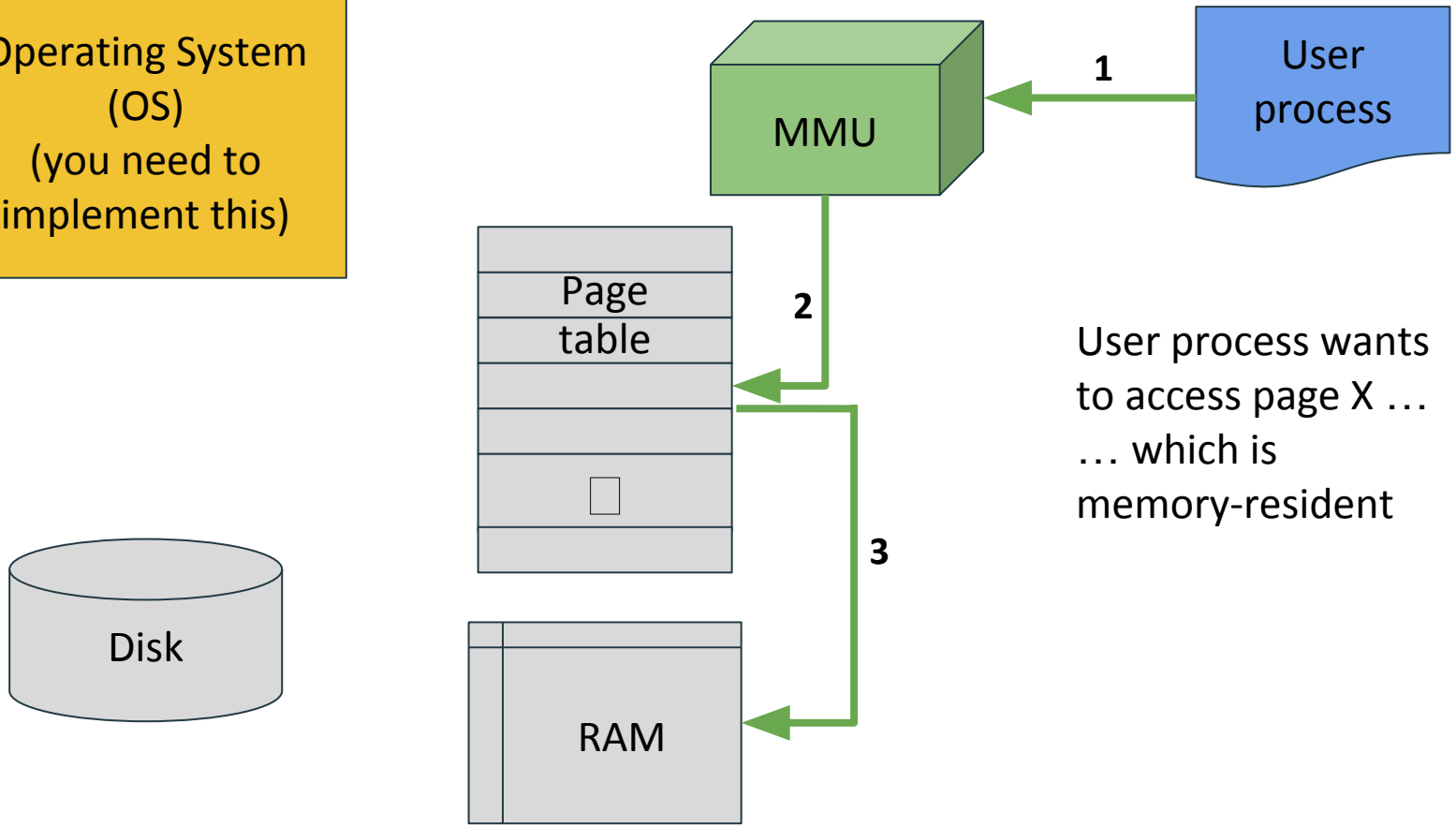
- Implement a (simulated) page fault handler that uses the second-chance page replacement algorithm
- Implement dynamic page allocation/deallocation (mmap/munmap)

Accessing Page X: General Steps



1. Check **page table**:
   - Is page **X *memory resident***?
     - Yes: Access physical memory location. Done.
     - No: Continue to the next step
2. **Page Fault: Trap** to OS (OS is informed)
3. Locate page X in secondary storage
4. Load page X into a physical memory frame
5. Update page table
6. Go to step 1 to retry

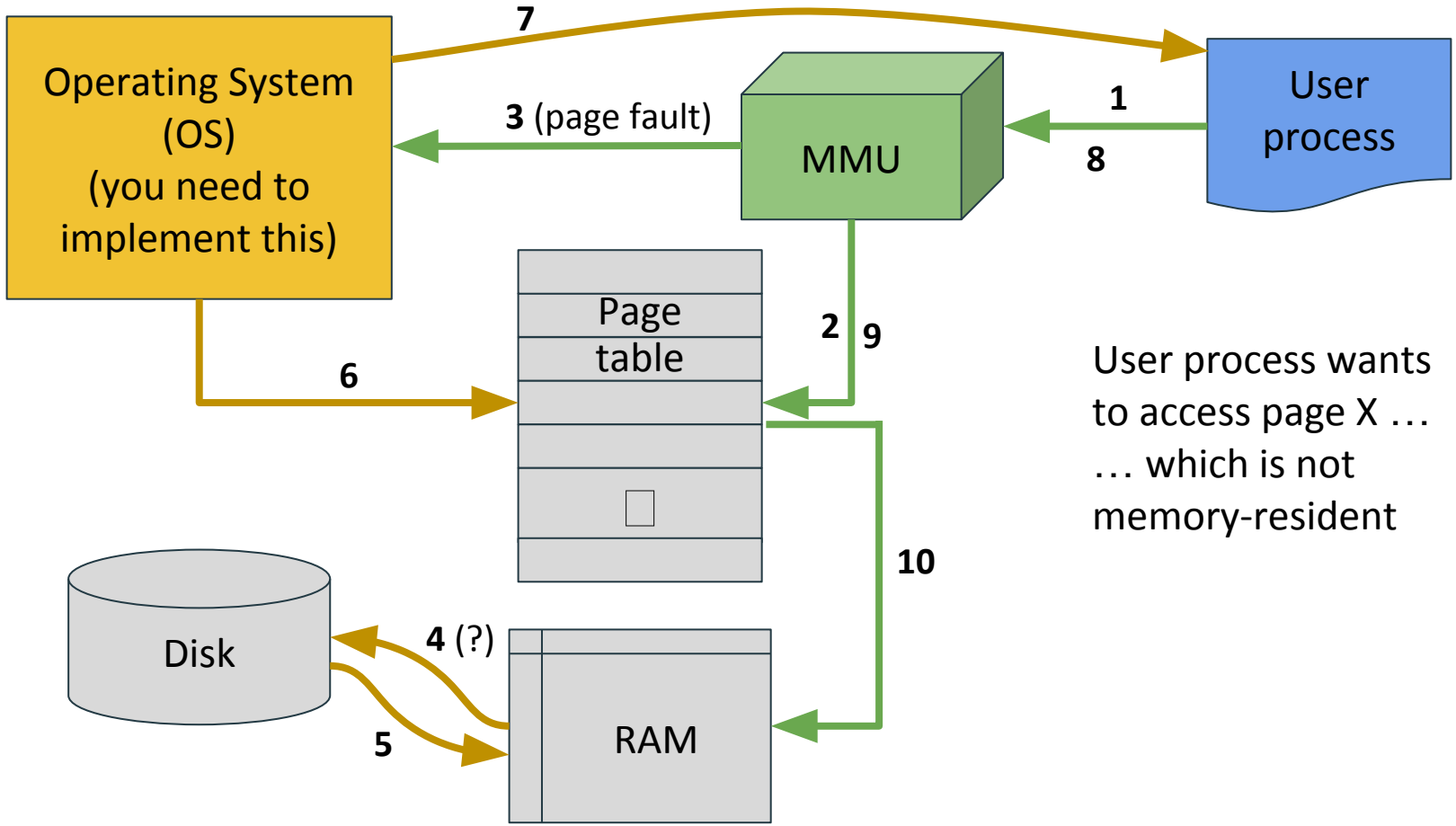
Operating System (OS) (you need to implement this)



User process wants to access page X …
… which is memory-resident

Steps for memory-resident page

1. User process tries to access page X
2. MMU checks the page table for page X, and finds that it is in frame Y in RAM
3. MMU reads/writes the data in frame Y



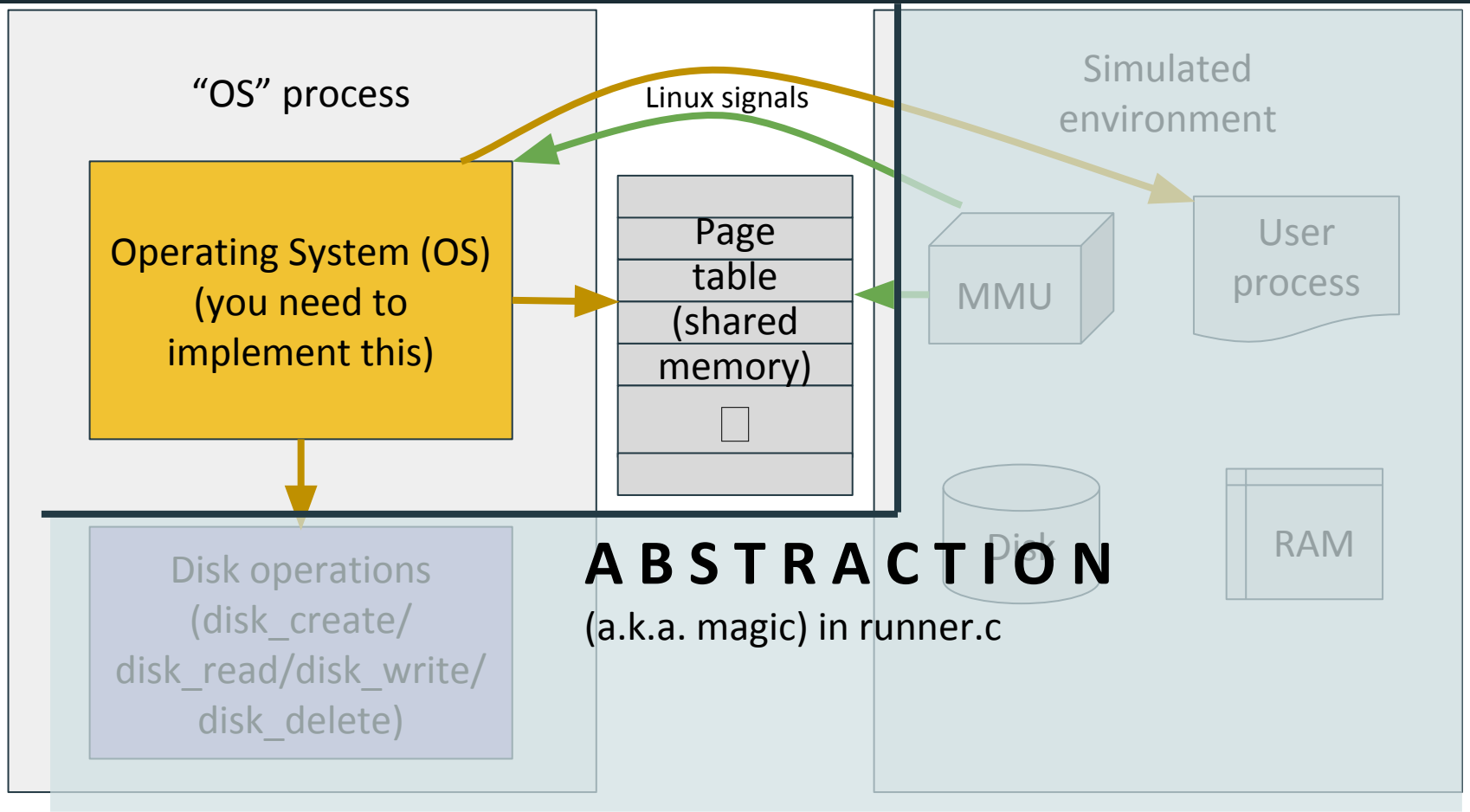
User process wants to access page X …
… which is not memory-resident

Steps for non memory-resident page

1. User process tries to access page X
2. MMU checks the page table for page X, and finds that it is not in RAM
3. MMU triggers a page fault
4. OS writes evicted page from RAM to disk if page is dirty
5. OS loads new page from disk to RAM

Steps for non memory-resident page

6. OS updates page table
7. OS resumes the user process
8. User process tries to access page X (again)
9. MMU checks the page table for page X (again), and finds that it is in frame Y in RAM
10. MMU reads/writes the data in frame Y


"OS" process
Linux signals
Simulated environment
Operating System (OS) (you need to implement this)
Page table (shared memory)
MMU
User process
Disk operations (disk_create/ disk_read/disk_write/ disk_delete)
Disk
RAM
ABSTRACTION
(a.k.a. magic) in runner.c

Exercise 1: Second chance page replacement

- Implementation of second chance page replacement algorithm with read-only pages
  (No need to write any pages from RAM to disk)
- Use disk_read() to load a page from disk to RAM

- Fret not about the length of the write-up; it has been said that “once you understand ex1, you are done with the assignment” :)

Exercise 1: Second chance page replacement

- This exercise is meant to check that your implementation adheres to the deterministic behaviour specified in the write-up
  - Each page will end up in a specific frame if the algorithm is followed exactly (i.e. the disk_read() calls must match ours exactly)
  - Note: There is a slight difference in the MMU between this assignment and the lecture: In this assignment, after a page fault, the MMU will set the referenced bit on the page freshly brought to RAM from the disk

Exercise 2: Ex1 + write to disk + segfault

- If the to-be-evicted page is dirty, write the page back to disk using disk_write()
- If the requested page does not exist, your OS should detect it and reply with the appropriate signal

Exercise 3: Ex2 + mmap/munmap

- Listen to a different signal for page allocation / deallocation requests
  - This mimics mmap()/munmap() behaviour, but the user program will only make requests for a single page at a time
  - When allocating a page, remember that you have to call disk_create() before disk_read() can be called
  - When deallocating a page, you should call disk_delete() if that page currently exists on the disk

Exercise 3: Ex2 + mmap/munmap

- When allocating a page, your OS can choose any available page to allocate
  - Any valid choice of page will be accepted when grading
  - (Note: The page number given to your OS may not be the same as the number specified in the input file; read the write-up for details)
- Efficiency considerations
  - mmap()/munmap() should have time complexity independent of the number of pages and frames (i.e. mmap/munmap should not become slower when there are many pages and frames)

Exercise 4: Ex3 + lazy creation of pages on disk

- This is a bonus exercise
- disk_create() should not be called on initialisation and on mmap; instead it should be called at the first time the user process tries to access that page

General notes

- You should only submit ex1.c/ex2.c/ex3.c/ex4.c
  - Note: Even though ex1 is the demo exercise, the submission of ex1.c constitutes 1 out of the 2 marks for ex1
- We may replace runner.c and change the constants in page_table.h when grading

For exercise 1 demo

- Run your code with ex1_sample1.in, and show me the output
- Answer any other questions

Questions?