# Programming Homework Assignment #3

J. H. Wang

Apr. 24, 2023

## Programming Homework #3

- Programming exercises
  - Programming problems: 7.17*, 8.25*, 9.26*
    - Note: Each student must complete all programming problems on your own
  - Programming projects for Chap. 7* and Chap. 9*
    - Team-based: At least one selected programming project
- Due: three weeks (May 15, 2023)

# Programming Problems

- 7.17*: Implement your solution to Exercise 7.15 using POSIX synchronization (or Java if you want).
- In particular, represent northbound and southbound farmers as separate threads.
- Once a farmer is on the bridge, the associated thread will sleep for a random period of time, representing traveling across the bridge.
- Design your program so that you can create several threads representing the northbound and southbound farmers.

# (Exercise 7.15 for your reference)

- A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge.
- Farmers in the two villages use this bridge to deliver their produce to the neighboring town.
- The bridge can become deadlocked if a northbound and a southbound farmer get on the bridge at the same time. (Vermont farmers are stubborn and are unable to back up.)
- Using semaphores and/or mutex locks, design an algorithm in pseudocode that prevents deadlock.
- Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).

- 8.25*: Assume that a system has a 32-bit virtual address with a 4-KB page size. Write a program that is passed a virtual address (in decimal) on the command line and have it output the page number and offset for the given address.

  (to be continued…)

– (… continued from the previous slide)

As an example, your program would run as follows:

```
./a.out 19986
```

Your program would output:

```
The address 19986 contains:
page number=4
offset=3602
```

Writing this program will require using the appropriate data type to store 32 bits. We encourage you to use unsigned data types as well.

– 9.26*: Write a program that implements the FIFO, LRU, and optimal page-replacement algorithms presented in this chapter. First, generate a random page-reference string where page numbers range from 0 to 9. Apply the random page-reference string to each algorithm, and record the number of page faults incurred by each algorithm. Implement the replacement algorithm so that the number of page frames can vary from 1 to 7. Assume that demand paging is used.

## Programming projects – Choose one from Ch.7 or Ch.9

- Ch.7*: Banker's Algorithm
  - Write a multithreaded program that implements the *banker's algorithm* discussed in Section 7.5.3. This assignment combines three topics: (1) multithreading (2) preventing race conditions (3) deadlock avoidance.
  - Create n customer threads that request and release resources from the bank. The customers will continually loop, requesting and then releasing random numbers of resources. The banker will grant a request if it satisfies the safety algorithm.
  - Since multiple threads will concurrently access shared data, access must be controlled through mutex locks to prevent race conditions.
  - You should invoke your program by passing the number of resources of each type on the command line.

- Ch.9*: Designing a Virtual Memory Manager
  - This project consists of writing a program that translates logical to physical addresses for a virtual address space of size $2^{16}$=65,536 bytes.
  - Your program will read from a file containing logical addresses and, using a TLB as well as a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address.

- The test file *addresses.txt* contains several 32-bit integer numbers that represent logical addresses. However, you need only be concerned with 16-bit addresses, so you must mask the rightmost 16 bits of each logical addresses. These 16 bits are divided into (1) an 8-bit page number and (2) 8-bit page offset.
- Your program will implement demand paging. The backing store is represented by the file BACKING_STORE.bin, a binary file of size 65,536 bytes

## Any Questions or Comments?