# Homework Assignment #2

J. H. Wang

Apr. 10, 2023

# Homework #2

- Chap.4: 4.1, 4.3, 4.4
- Chap.5: 5.2, 5.8, 5.14, 5.15
- Chap.6: 6.4, 6.6, 6.10
- Programming exercises:
  - Programming problems: 4.17*, (4.21**, ) 6.33*
    - Note: Each student must complete all programming problems on your own
  - Programming projects for Chap. 4* (*2) & Chap. 6* (*3)
    - Team-based: At least one selected programming project from each chapter
- Due: two weeks (Apr. 24, 2023)

- Chap. 4
  - 4.1: Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.
  - 4.3: Which of the following components of program state are shared across threads in a multithreaded process?
    - (a) Register values
    - (b) Heap memory
    - (c) Global variables
    - (d) Stack memory

- 4.4: Can a multithreaded solution using multiple *user*-level threads achieve better performance on a multiprocessor system than on a single-processor system? Explain.

- Chap. 5:
  - 5.2: Discuss how the following pairs of scheduling criteria conflict in certain settings.

    (a) CPU utilization and response time
    (b) Average turnaround time and maximum waiting time
    (c) I/O device utilization and CPU utilization

– 5.8: The following processes are being scheduled using a preemptive round-robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as Pidle). This task has priority 0 and is scheduled whenever the system has no other available processes to run.

(… to be continued)

- The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

| Thread | Priority | Burst | Arrival |
|---|---|---|---|
| $P_1$ | 40 | 20 | 0 |
| $P_2$ | 30 | 25 | 25 |
| $P_3$ | 30 | 25 | 30 |
| $P_4$ | 35 | 15 | 60 |
| $P_5$ | 5 | 10 | 100 |
| $P_6$ | 10 | 10 | 105 |

(… to be continued)

- (a) Show the scheduling order of the processes using a Gantt chart.
  (b) What is the turnaround time for each process?
  (c) What is the waiting time for each process?
  (d) What is the CPU utilization rate?

- 5.14: Consider a preemptive priority scheduling algorithm based on dynamically changing priorities.
  - Larger priority numbers imply higher priority
  - When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate $\alpha$
  - When it is running, its priority changes at a rate $\beta$
  - All processes are given a priority of 0 when they enter the ready queue
  - The parameters $\alpha$ and $\beta$ can be set to give many different scheduling algorithms

(a) What is the algorithm that results from $\beta > \alpha > 0$?

(b) What is the algorithm that results from $\alpha < \beta < 0$?

– 5.15: Explain the differences in how much the following scheduling algorithms discriminate in favor of short processes:
(a) FCFS
(b) RR
(c) Multilevel feedback queues

- Chap.6:
  - 6.4: Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.
  - 6.6: The Linux kernel has a policy that a process cannot hold a spinlock while attempting to acquire a semaphore. Explain why this policy is in place.

- 6.10: The implementation of mutex locks provided in Section 6.5 suffers from busy waiting.
- Describe what changes would be necessary so that a process waiting to acquire a mutex lock would be blocked and placed into a waiting queue until the lock became available.

# Programming Problems

- 4.17*: An interesting way of calculating pi is to use a technique known as *Monte Carlo*, which involves randomization. This technique works as follows:
  - Suppose you have a circle inscribed within a square, (Assume that the radius of this circle is 1.)
  - First, generate a series of random points as simple (x,y) coordinates
  - These points must fall within the Cartesian coordinates that bound the square
  - Of the total number of random points that are generated, some will occur within the circle
  - (… to be continued)

    - Next, estimate pi by performing the following calculation:
        - Pi=4*(number of points in circle) / (total number of points)
- Write a multithreaded version of this algorithm that creates a separate thread to generate a number of random points.
    - The thread will count the number of points that occur within the circle and store that result in a global variable.
    - When this thread has exited, the parent thread will calculate and output the estimated value of pi.

- [optional] (4.21**): The *Fibonacci sequence* is the series of numbers 0, 1, 1, 2, 3, 5, 8, …. .
  Formally, it can be expressed as:
  $fib_0=0$
  $fib_1=1$
  $fib_n=fib_{n-1}+fib_{n-2}$
- Write a multithreaded program that generates the Fibonacci sequence using either the Java, Pthread, or Win32 thread library.
  (… to be continued)

- This program should work as follows:
  - On the command line, the user will enter the number of Fibonacci numbers that the program is to generate
  - The program will then create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that can be shared by the threads (an array is probably the most convenient data structure)
  - When the thread finishes execution, the parent thread will output the sequence generated by the child thread
  - Because the parent thread cannot begin outputting until the child finishes, the parent will have to wait for the child thread to finish

- 6.33*: Modify the program in Exercise 4.17 so that you create several threads, each of which generates random points and determines if the points fall within the circle.
  - Each thread will have to update the global count of all points that fall within the circle.
  - Protect against race conditions on updates to the shared global variable by using mutex locks.

  - (Note: You can use mutex lock or semaphores in Pthread, or Windows API if you want.)

# End-of-Chapter Programming Projects

- Programming Projects for Chap. 4: (Choose one)
  - Project 1. Sudoku solution validator
    - To design a multithreaded application that determines whether the solution to a Sudoku puzzle is valid.
    - Passing parameters to each thread
    - Returning results to the parent thread
  - Project 2. Multithreaded sorting application
    - Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads sort each sublist using a sorting algorithm of your choice. The two sublists are then merged by a third thread.

# End-of-Chapter Programming Projects

- Programming Projects for Chap. 6: (Choose one)
  - Project 1: The Sleeping Teaching Assistant
    - Room: 1 desk with a chair and computer
    - Hallway: 3 chairs
    - POSIX threads, mutex locks, and semaphores
  - Project 2: The Dining Philosophers Problem
    - Pthread mutex locks and condition variables
  - Project 3: Producer-Consumer Problem
    - Use standard counting semaphores for empty and full and a mutex lock, rather than a binary semaphore, to represent mutex.
    - Producer and consumer threads
    - Pthreads mutex locks/semaphores
    - Windows mutex locks/semaphores

# Homework Submission

- For hand-written exercises, please hand in your homework on paper in class
- For programming exercises, please upload your program to the iSchool+ as follows:
  - Program uploading: a compressed file (in .zip format) including source codes, execution snapshot, and documentation
  - The documentation should clearly identify:
    - Team members and responsibility
    - Compilation or configuration instructions if it needs special environment to compile or run
  - Please contact with the TA if you are unable to upload your homework

## Any Question or Comments?