Homework03

Ameya Mellacheruvu, Clarissa Bernardo, and Krysten Tachiyama

1. Write an implementation of the Dining Philosophers program, demonstrating deadlock avoidance.

   See file called diningPhilosphers.c

2. Write a short paragraph explaining why your program is immune to deadlock?

   Referenced source:
   https://medium.com/swlh/the-dining-philosophers-problem-solution-in-c-90e2593f64e8

   We used semaphores to avoid deadlock. We used 2 types of semaphores: 1) binary - accommodates the chopsticks. Binary is used when there is only one instance of that resource. Since the chopsticks are the shared resource and philosophers have to wait for them in order to eat, binary type semaphores are most appropriate. 2) counting - accommodates the room. We have 4 instances of that since we can have 4 philosophers eating at the same time. Semaphores are special types of variables that can control the access of a shared resource. The built in methods in the semaphore.h library allows the program to be immune to deadlock. The sem_wait() method checks if the resource is available before allowing the resource to be allocated to the philosopher. This queue prevents philosophers from entering all at once and "grabbing" chopsticks since there are only 4 spots that can be occupied.

3. Modify the file-processes.cpp program

   See fileProcess.c

4. Write a program that opens a file in read-only mode and maps the entire file into the virtual-memory address space using mmap. The program should search through the bytes in the mapped region, testing whether any of them is equal to the character X. As soon as an X is found, the program should print a success message and exit. If the entire file is searched without finding an X, the program should report failure. Time your program on files of varying size, some of which have an X at the beginning, while others have an X only at the end or not at all.

   See file mmap.c

   Observations:

- tested very small file with an X - took .088 milliseconds
- tested bigger file with an X - too a little longer - .187 milliseconds
- tested the bigger file with X within the first 5 characters - .114 milliseconds.
- iterating through bigger files will take longer.

5. Read enough of Chapter 10 to understand the following description: In the TopicServer implementation shown in Figures 10.9 and 10.10 on pages 456 and 457, the receive method invokes each subscriber's receive method. This means the TopicServer's receive method will not return to its caller until after all of the subscribers have received the message. Consider an alternative version of the TopicServer, in which the receive method simply places the message into a temporary holding area and hence can quickly return to its caller. Meanwhile, a separate thread running in the TopicServer repeatedly loops, retrieving messages from the holding area and sending each in turn to the subscribers. What Java class from Chapter 4 would be appropriate to use for the holding area? Describe the pattern of synchronization provided by that class in terms that are specific to this particular application.

   The bounded buffer class is appropriate for the holding area because it allows the producer to store values into the intermediate storage, or buffer. The consumer can then retrieve the value from the buffer when it is ready.

   With the Topic Server example, the receive method stores the message into the buffer and the other thread sends the message to the correct subscribers. This allows the receiver thread to bring and store messages while the other thread consumes and sends them.