1) (Concurrency and Synchronization)

The Starbucharest espresso franchise serves customers FIFO in the following way. Each customer entering the shop takes a single “ticket” with a number from a “sequencer” on the counter. The ticket numbers dispensed by the sequencer are guaranteed to be unique and sequentially increasing. When a barrista is ready to serve the next customer, he or she calls out the “service count”, the lowest unserved number previously dispensed by the sequencer. Each customer waits until the service count reaches the number on their ticket. Each barrista waits until the customer with the ticket number called places an order.

Show how to implement sequencers, tickets, and service counts using locks and condition variables. Your solution should include code for the barrista and customer threads.
2) (Scheduling)
Consider a system with two processes, \( H \), a high priority process, and \( L \), a low priority process. The scheduler chooses the highest priority ready process to be the one to run at each scheduling opportunity. Suppose that \( L \) has just run and entered a critical section protected by a spin-lock (a busy-waiting implementation). Now \( H \) becomes ready to run, having completed some I/O that previously had it blocked, and is scheduled immediately. Then \( H \) tries to enter its critical section by trying to acquire the spin-lock.

a. What is the resulting behavior with respect to \( H \) and \( L \)?

b. What would the behavior be if the scheduler used a round robin policy?

c. What would the behavior be if this were a multiprocessor using priority scheduling on a common ready queue?

d. Suppose a blocking mutex lock were used instead of a spin-lock? In that case, what is the behavior of priority scheduling (consider the existence of processes beside just \( H \) and \( L \)?)
3) (Potential pitfalls in inter process communication) A distributed system using mailboxes has two IPC primitives, SEND and RECEIVE. The latter primitive can specify a process from which a message is expected and blocks the calling process if no message from that designated sending process is available, even though messages may be waiting from other processes. Processes need to communicate frequently, but are not using this communication to explicitly synchronize access to any shared resources. Discuss whether deadlock is possible in this scenario.

In another version of message-passing with SEND and RECEIVE, there is no blocking when attempts are made to send to a full mailbox or trying to receive from an empty one. These cases return an error code and the process can respond by just trying again, over and over until it succeeds. In this scenario, assume processes are not being picky about which process sent a message if one is available. Discuss briefly whether this scenario leads to any undesirable behavior.
4) (Memory Locality) Assume that the total amount of real physical memory in a computer system is fixed. Give examples of either code fragments or page reference strings that exhibit the behaviors below. In each case, give an *intuitive explanation* (25 words or less) for the behavior shown. A drawing is often helpful in this explanation.

(a) Doubling the page size (thus halving the number of frames available) can reduce page faults.

(b) Halving the page size (thus doubling the number of frames available) can reduce page faults.
5) (Concurrency)
Consider the following incorrect solution to the critical section problem. There are two threads, P0 and P1, and they share a Boolean 2-element array, flag[2], both elements initially FALSE, and an integer variable, turn initially 0.

P0 code

a) while (TRUE) {
  b) flag[0] = TRUE;
  c) while (turn != 0) {
     d) while (flag[1])
     e) {no_op}
     f) turn = 0;
    }
  g) critical section;
  h) flag[0] = FALSE;
     other stuff;
}

P1 code

A) while (TRUE) {
 B) flag[1] = TRUE;
 C) while (turn != 1) {
 D) while (flag[0])
 E) {no_op}
 F) turn = 1;
  }
 G) critical section;
 H) flag[1] = FALSE;
     other stuff;
}

Give an interleaving of the two threads' execution (e.g. a A b B c C ... ) that shows that the solution violates the correctness conditions of the critical section problem.