Synchronization Tools and Examples (Chapters 6 and 7)

Operating Systems Comp Sci 3SH3, Winter 2024

Prof. Neerja Mhaskar

updating a value whether balance += value or balance -= value is not an atomic operation. Balance is loaded in a register, the register is added or subtracted with the value and the register overwrites the balance.

1. Consider a banking system that maintains an account balance with two functions: `deposit(amount)` and `withdraw(amount)`. These two functions are passed the amount that is to be deposited or withdrawn from the bank account. Assume that a husband and wife share a bank account. Concurrently, the husband calls the `withdraw()` function and the wife calls `deposit()`. Describe with an example how a race condition is possible and what might be done to prevent the race condition from occurring.

2. What are the three solution requirements to the critical section (CS) problem. Explain.

   Mutual Exclusion, Progress and Bounded Waiting

3. Prove that the following critical solution problem fails. Also state explicitly state which of the critical section problem solution requirements are not satisfied and why.
   Note: We assume that load and store machine language instructions are atomic.

   The two processes share the following variable:
   `int turn;`
   a. **`turn`** indicates whose turn it is to enter the critical section.
   b. You can initialize `turn = 0` or `turn = 1`. The failure of the solution is independent of the initialization.

   Algorithm for process P_i

```
do {
      while (turn==j);
      critical section
      turn = j;
      remainder section
} while (true);
```

4. Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism—a spinlock or a mutex lock where waiting processes sleep while waiting for the lock to become available:
   a. The lock is to be held for a short duration.
   b. The lock is to be held for a long duration.
   c. A thread may be put to sleep while holding the lock.

5. Illustrate how you would use semaphores to synchronize processes P1, P2, P3, P4 where P1 is executed before P2, P2 is executed before P3, P3 is executed before P4. Write pseudocode that Illustrates how a binary semaphore can be used to implement mutual exclusion among *n* processes.

6. Consider two concurrently running processes ***P_1*** and ***P_2*** that require ***P_1*** to execute before ***P_2*** (here ***P_1*** and ***P_2*** may or may not access/modify shared variables.). Use semaphores to synchronize processes ***P_1*** and ***P_2***.

7. Illustrate how you would use semaphores to synchronize processes P_1, P_2, P_3, P_4 where P_1 is executed before P_2, P_2 is executed before P_3, P_3 is executed before P_4.

8. What is the difference between a semaphore and conditional variable?

9. Design an algorithm for a bounded-buffer monitor in which the buffers (portions) are embedded within the monitor itself.

10. In the solution for dining philosophers' problem using monitors, provide a scenario in which a philosopher may starve to death.

11. Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.

12. Implement a mutex lock using the `test_and_set()` atomic hardware instruction. Assume that the following structure defining the mutex lock is available:

```
typedef struct {
bool held;
} lock;
```

`held == false (true)` indicates that the lock is available (not available) Using the `struct lock`, illustrate how the following functions can be implemented using the `test_and_set()` instructions:

   a. `void acquire(lock *mutex)`
   b. `void release(lock *mutex)`

Be sure to include any initialization that may be necessary.