Operating System: Synchronization

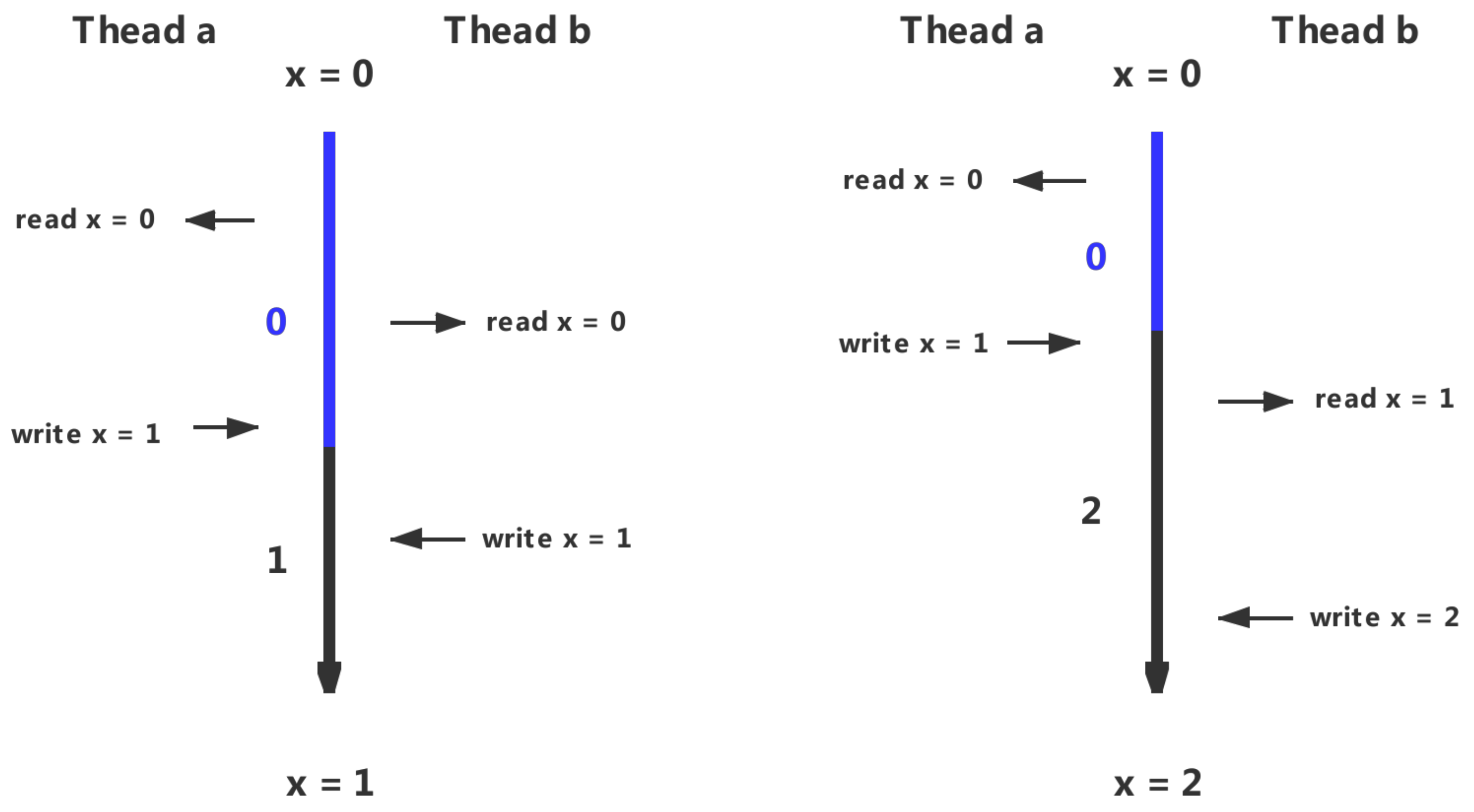
LI Penghui s1155137827

Explanations

- Modern operating systems’ support for synchronized access to shared objects in multi-threaded programs
    - Multi-threaded programs
    - Shared objects
    - Synchronized access

Synchronization Motivation

- Program behaviour is undefined when threads concurrently read/write shared memory
  - Thread a & b concurrently execute code **x ++** on shared variable **x;**



Synchronization Motivation

- Program behaviour is undefined when threads concurrently read/write shared memory
- Program execution is nondeterministic
    - Thread scheduler makes different decisions
- Compiler and architecture reorder instructions
- Race condition
    - Behaviour of program depends on the interleaving of operations of different threads

Synchronization Variables

- Lock
    - Enables mutual exclusion by providing two methods: Lock:: acquire() & Lock:: release()
    - Has two states: BUSY or FREE
    - Initially in FREE state
- Condition variable
    - Enables a thread to efficiently wait for a change to shared state protected by a lock
    - CV:: wait(Lock *lock), CV:: signal(), CV:: broadcast()

Detail: Lock

- Lock:: acquire()
    - Wait until lock is free, then take it
- Lock:: release()
    - Release lock, wake up anyone waiting for it
- At most one thread holds a lock

Case Study: Bank Account Object

- A bank account object includes a list of transactions and a total balance.

- To add a new transaction

  - Acquire the account's lock, append the new transaction, read the old balance, write a new balance, and release the lock

- To query the balance and list of recent transactions

  - Acquire the account's lock, read recent transactions, read the balance, and release the lock

Detail: Condition Variables

- CV:: wait(Lock *lock)
  - Atomically releases the lock and suspends execution of the calling thread, placing it onto the condition variable’s waiting queue
- CV:: signal()
  - Awake one waiting thread off the waiting queue
- CV:: broadcast()
  - Awake all waiting threads off the waiting queue

Case Study: Bounded Queue w/ CV

```
get(  ){
      lock.acquire();
      while(fornt = tail){
            empty.wait(lock);
      }
      item = buf[front % MAX];
      front ++;
      full.signal(lock);
      lock.release();
      return item;
}
```

```
put(item){
      lock.acquire();
      while((tail - front == MAX)){
            full.wait(lock);
      }
      buf[tail % MAX] = item;
      tail ++;
      empty.signal();
      lock.release()
}
```

- Initially: front = tail = 0; MAX is buffer size
- Empty & full are condition variables

Conclusion

- Shared objects among threads make multi-threaded programs vastly simpler
- Ensure the safety and correctness of program execution
- However, synchronization also brings problem if not correctly used
    - deadlock: a set of members are blocked because each member is holding a resource and waiting for another resource acquired by someone else.

Operating System: Synchronization

- Thanks
- Q & A