1. a) Trap instruction is run in user mode, and privileged operation is run in kernel mode

Notes

- **Previliged Instructions**
  - Is the instruction that can run only in **kernel mode**
  - Attempt at execution in **user mode** → treated as an illegal operation & will not run.
- **Trap**
  - Is a special hardware instruction
  - Is a software generated interrupt [4]
  - Is a type of synchronous interrupt [1]
  - Is caused by an exceptional condition [1]
    1. Division by zero [1]
    2. Invalid memory access (segmentation fault) [1]
    3. Previleged instruction by **user mode** code [2]
  - Usually results in a switch to **kernel mode** → Operating system performs action → Returns control to oroginal process
- **Trap Instruction**
  - Is executed when a user wants to invoke a service from the operating system (i.e. reading hard drive) in **user mode**
- **User Mode**
  - Executing code has no ability to *directly* access hardware or reference memory [3]
  - Crashes are always recoverable [3]
  - Is where most of the code on our computer are executed [3]
- **Kernel Mode**
  - Executing code has complete and unrestricted access to the underlying hardware [3]
  - Is generally reserved for the lowest-level, most trusted functions of the operating system [3]
  - Is fatal to crash; it will halt the entire PC (i.e the blue screen of death) [3]

References

1) Wikipedia, Trap (computing), link
2) University of Utah, CS5460: Operating Systems Lecture 3 - OS Organization, link
3) Coding Horror, Understanding User and Kernel Mode, link
4) ETH Zurich, Programming in Systems, link

b) No. Lock uses a variable with binary states 0 (acquired) and 1 (available), where as semaphore uses counter variable that can have value greater than 1 to keep track of the amount of resource remaining.

Notes

- **Locks**
  - Is a variable with two boolean states
    - 1 - (available/unlock/free)
    - 0 - (acquired/locked/held)
  - Has two operations
    1. `acquire()`

```
boolean test_and_set(boolean *lock)
{
      boolean old = *lock;
      *lock = True;
      return old;
}
boolean lock;

void acquire(boolean *lock) {
       while(test_and_set(lock));
}
```

    2. `release()`

```
void release(boolean *lock) {
       *lock = false;
}
```

  - Is put around critical section to ensure critical section executes as if it's a single atomic instruction

```
1  lock_t mutex; // some globally-allocated lock 'mutex'
2  ...
3  lock(&mutex);
4  balance = balance + 1;
5  unlock(&mutex);
```

  - Can only be released by the thread that acquired it
  - Is used to protect shared resource (e.g. from race condition in files and data structure) [2]
- **Semaphore**
  - Is an abstract data types suitable for synchronization problems [2]

- Has variable `count` that allows arbitrary resource count [1]
- Has two atomic operations
  1. (`wait/P/decrement`) - block until `count > 0` then decrement variable

```
wait(semaphore *s) {
        while (s->count == 0) ;
        s->count -= 1;
}
```

  2. (`signal/V/increment`) - increment `count`, unblock a waiting thread

```
signal(semaphore *s) {
        s->count += 1;
        ......                  //unblock one waiter
}
```

- Can be signaled by any thread [2]

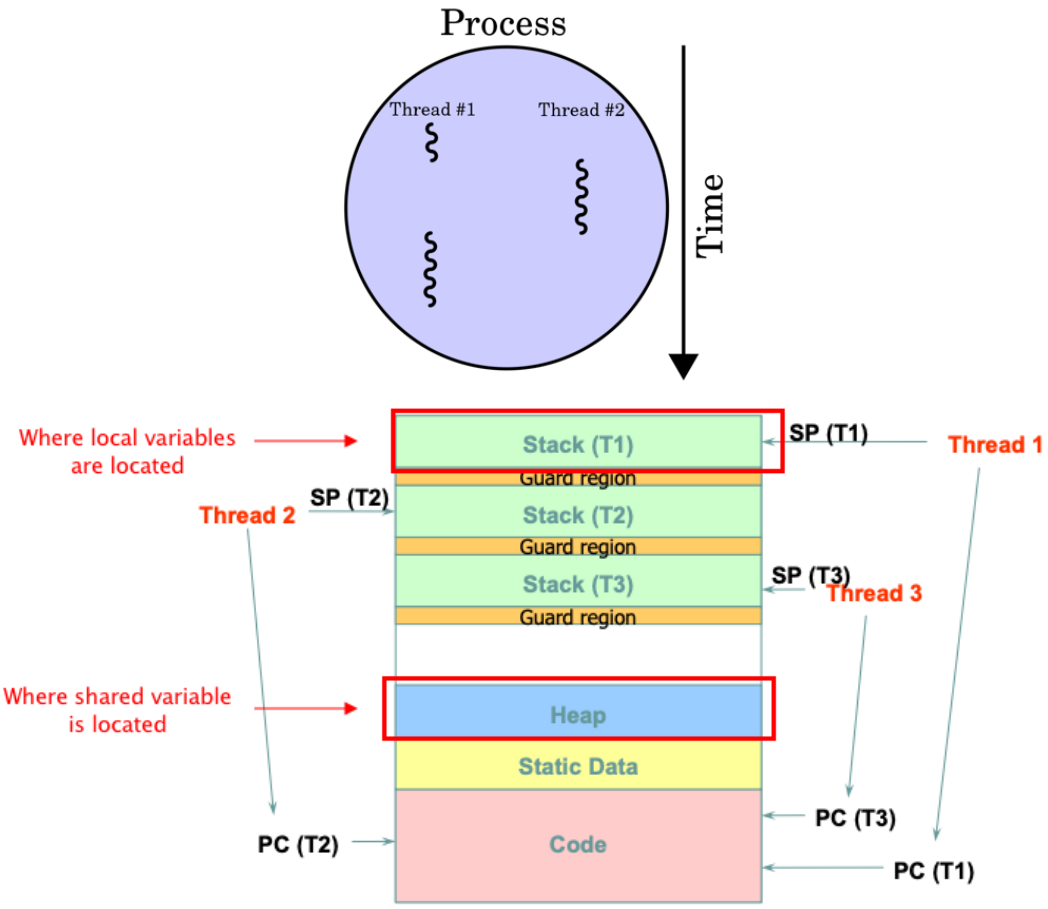
References

1) Wikipedia, Semaphore (programming), link
2) Stack Overflow, Difference between binary semaphore and mutex, link

c) All shared variable of threads are located in heap memory, and if multiple threads access the variable at the same time, a race condition would occur.

Notes

- What is concurrency error? Where and when does it occur?
- **Race condition**
  - if multiple threads of execution enter the critical section roughly at the same time; both attempt to update the shared data structure, leading to a supring result
- **Concurrency**
  - Is the ability of different parts or units of a program, algorithm, or problem to be executed out of order, without affecting the final outcome. [1]
- **Concurrency Error**
  - Two types of concurrency errors
- **Thread**
  - Is the smallest sequence of programmed instructions that can be managed independently by a schdeduler [2]



– A thread is bound to a single process
– A process can have multiple threads

References

1) Wikipedia, Concurrency (computer science), link
2) Wikipedia, Thread, link