Michael Laucella
HW 3

1.

a) the code will defy expectations, fputs will either output proc_info or a nullptr. This is because there is no gaurantee that after the if() proc_info cannot be changed to nullptr before the fputs call

b) 1) use a lock and surround each section

```
lock();                    lock();
if( condition)             value = nullptr;
  fputs(value)             unlock();
unlock();
```

2) [redacted]

(it's attached later)

2.

a) Sentinel's allow for decoupling of enqueue and dequeue because H and T will never point to the same thing. Because of this general concurrency is easier because we can dequeue and enqueue at the same time, but also lock free structures because of the linearization point which allows enqueues and deques to have a deterministic valid state because of the tail pointer

1. b. 2.)

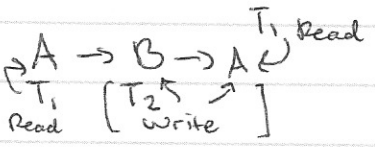
Make a copy of the pointer then check based on the copy

```
tmp_ptr = proc_info          proc_info = nullptr
if(tmp_ptr)
   fputs(tmp_ptr)
```

even if proc_info is set to a null pointer the information it was referencing wont dissapear hence accessing the tmp_ptr.

3.

The ABA problem is when a value has been read from a thread, then changed by another, and reread again and not realizing a change [and changed back] ever occurred



Hardware would require X86/X64 so that CAS operations can be used
then with CAS modify the lower bits of the pointer to denote the number of changes

4.

linearization creates a state of certainty, it is the dividing line between two states when a task is or is not complete and what will be available to the whole process which will be after the linearization point,

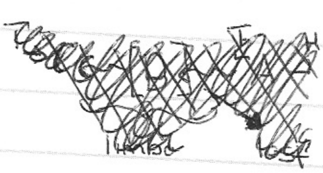
for instance in the CAS queue the linearization point is at the tail, until the tail pointer is reassigned the new node being created is not visible to the head pointer

5.

a) pros - no locking, good for structures which have points of high impact (will be used frequently ex: (Stack/Queue/list insert)

cons - difficult to reason about/create
difficult to debug
requires specific hardware (x86/x64)
not appropriate for all types of problems
better for larger systems

b) the tail pointer needs to be either the last or second to last entry in the queue,
obviously if an entry succeeds from start to end then the tail will point to the end, however if it passes (CAS 1 (curr-tail->next == tail->next) tail->next = node)
then the tails next has been reassigned however if ~~[illegible]~~ it fails CAS 2 then that node is still there and pointed to by the previous node but the tail needs to be fixed. This is fine however ~~[illegible]~~ it
~~it [illegible] beyond the second to last entry then [illegible] node would be last upon fixing the tail, [illegible]~~



cannot extend beyond this because the other enqueues will attempt to reassign tail next, if tail has not moved this cannot happen again making the tail at most point to the second to last element