Hardware Realization of Lock Instruction

```
boolean TestAndSet(boolean *target) {
   boolean rv = *target;
   *target = TRUE;
   return rv;
}
```

```
do {
   while (TestAndSetLock(&lock))
      ; // do nothing

      // critical section

   lock = FALSE;

      // remainder section
}while (TRUE);
```

Alternate Realization of Lock Instruction

```
int compare_and_swap(int *value, int expected, int new_value) {
  int temp = *value;

  if (*value == expected)
    *value = new_value;

  return temp;
}
```

```
do {
  while (compare_and_swap(&lock, 0, 1) != 0)
    ; /* do nothing */

    /* critical section */

  lock = 0;

    /* remainder section */
} while (true);
```

Bounded-waiting mutual exclusion with TestAndSet( ).

```
do {
   waiting[i] = TRUE;
   key = TRUE;
   while (waiting[i] && key)
      key = TestAndSet(&lock);
   waiting[i] = FALSE;

      // critical section

   j = (i + 1) % n;
   while ((j != i) && !waiting[j])
      j = (j + 1) % n;

   if (j == i)
      lock = FALSE;
   else
      waiting[j] = FALSE;

      // remainder section
}while (TRUE);
```

- The key feature of the above algorithm is that a process blocks on the AND of the critical section being locked and that this process is in the waiting state.
- **On Critical Section Exit**, process does not just unlock the critical section and let the other processes have a free-for-all trying to get in.
- **But searches in an orderly progres**sion (starting with the next process on the list) for a process that has been waiting, and if it finds one, then it releases that particular process from its waiting state, without unlocking the critical section,
- thereby allowing a specific process into the critical section while continuing to block all the others.
- Only if there are no other processes currently waiting; is the general lock removed, allowing the next process to access to the critical section