P_0

T&S

0 (while loop breaks)

Exits ← Critical region

So if P_0 is preempted
no other process gets to access critical section

now P_1

T&S

1

exits ← Critical section

1/0 while loop breaks

The compare_and_swap Instruction

Definition

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

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

}
```

Properties

- Executed atomically
- Returns the original value of passed parameter value
- Set the variable value the value of the passed parameter new_value but only if *value == expected is true. That is, the swap takes place only under this condition.

Swap Instruction

Shared boolean variable lock initialized to false; each process has a local Boolean variable key

Solution:

```
do {
    key = True
    while (key == True):
    {
        Swap (&lock, &key);
    }
    // critical section
    key = false
    // reminder section
} while (TRUE);
```

Shared integer `lock` initialized to 0;
Solution:

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

    /* critical section */

    lock = 0;

    /* remainder section */
}
```

Does it solve the critical-section problem? YES

Bounded-waiting with compare-and-swap

```
while (true) {
    waiting[i] = true;
    key = 1;
    while (waiting[i] && key == 1)
        key = compare_and_swap(&lock,0,1);
    waiting[i] = false;
    /* critical section */
    j = (i + 1) % n;
    while ((j != i) && !waiting[j])
        j = (j + 1) % n;
    if (j == i)
        lock = 0;
    else
        waiting[j] = false;
    /* remainder section */
}
```