Αυστηρή εναλλαγή

(α)

```
while (TRUE) {
    while (turn != 0) /* βρόχος */;
    critical_region();
    turn = 1;
    noncritical_region();
}
```

(β)

```
while (TRUE) {
    while (turn != 1) /* βρόχος */;
    critical_region();
    turn = 0;
    noncritical_region();
}
```

TSL

```
enter_region:
    TSL REGISTER,LOCK

    CMP REGISTER,#0
    JNE enter_region

    RET

leave_region:
    MOVE LOCK,#0
    RET
```

Peterson

```
#define FALSE   0
#define TRUE    1
#define N       2

int turn;
int interested[N];

void enter_region(int process)
{
    int other;

    other = 1 - process;
    interested[process] = TRUE;
    turn = process;
    while (turn == process && interested[other] == TRUE)
}
```

```
void leave_region(int process)
{
    interested[process] = FALSE;
}
```

Producer/Consumer

```
#define N 100
int count = 0;
void producer(void)
{
    int item;
    while (TRUE) {
        item = produce_item();
        if (count == N) sleep();
        insert_item(item);
        count = count + 1;
        if (count == 1) wakeup(consumer);
    }
}

void consumer(void)
{
    int item;
    while (TRUE) {
        if (count == 0) sleep();
        item = remove_item();
        count = count - 1;
        if (count == N - 1) wakeup(producer);
        consume_item(item);
    }
}
```

Λύση με semaphores για το πρόβλημα Producer/Consumer

```
#define N 100
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;

void producer(void)
{
    int item;
    while (TRUE) {
        item = produce_item();
        down(&empty);
        down(&mutex);
        insert_item(item);
        up(&mutex);
        up(&full);
    }
}

void consumer(void)
{
    int item;
    while (TRUE) {
        down(&full);
        down(&mutex);
        item = remove_item();
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```