Peterson Algorithm:

```
int flag[2];
int turn;
void init() {
   flag[0] = flag[1] = 0;
   turn = 0;
}
void lock() {
   flag[self] = 1;
   turn = 1 - self;
    while ((flag[1-self] == 1) && (turn == 1 - self))
        ; //spin
}
void unlock() {
   flag[self] = 0;
}
```

Test and set:

```
typedef struct __lock_t {
    int flag;
} lock_t

void init(lock_t *lock) {
   lock->flag = 0;
}

void lock(lock_t *lock) {
    while (TestAndSet(&lock->flag, 1) == 1)
        ; // spin
}
void unlock(lock_t *lock) {
    lock->flag = 0;
}
```

Test and set with yield:

```
int flag;
void init () {
    flag = 0;
}

void lock() {
    while (test_and_set(&flag, 1) == 1)
        yield();
}

void unlock() {
    flag = 0;
}
```

Load Linked and Store Conditional:

```
int LoadLinked(int *ptr) {
    return *ptr;
}

int StoreConditional(int *ptr, int value) {
    if(no update to *ptr since LoadLinked to this
address){
        *ptr = value;
        return 1;
    } else {
        return 0;
    }
}

void lock(lock_t *lock) {
    while(1) {
        while(LoadLinked(&lock->flag) == 1)
            ; //spin
        if(StoreConditional(&lock->flag, 1) == 1)
            return;
     }
}

void unlock(lock_t *lock) {
    lock->flag = 0;
}
```

Ticket lock:

```
typedef struct __lock_t {
    int ticket;
    int turn;
} lock_t;
void lock_init(lock_t *lock) {
    lock->ticket = 0;
    lock->turn = 0;
}

void lock(lock_t *lock) {
    int myturn = fetch_and_add(&lock->ticket);
    while (lock->turn != myturn)
        ; // spin
}

void unlock(lock_t *lock) {
    lock->turn = lock->turn + 1;
}
```