Refactoring isBootStrap for Thread Safety

The Problem

ONElib.c line 282:

```
static bool isBootStrap = false;
```

This global static is modified during schema creation: - Set to `true` at line 292 (start of `oneSchemaCreateFromFile`) - Set to `false` at line 375 (end of `oneSchemaCreateFromFile`) - Used at line 1126 in `oneReadLine` (currently commented out)

Why it exists: During schema creation, the library must "bootstrap" itself by creating a minimal `OneFile` object that can read schema files, before the full schema infrastructure exists. The `isBootStrap` flag indicates this special state.

Thread safety issue: When two threads call `oneSchemaCreateFromFile()` concurrently:

```
Thread 1: isBootStrap = true  (start bootstrap)
Thread 2: isBootStrap = true  (overwrites!)
Thread 1: ... bootstrap operations ...
Thread 2: isBootStrap = false (ends early!)
Thread 1: ... WRONG STATE ...
Thread 1: isBootStrap = false (end)
```

Result: "multiple list types for linetype definition" errors.

Refactoring Options

Option 1: _Thread_local (Simplest) □

Change:

```
// Line 282
_Thread_local bool isBootStrap = false;
```

Pros: - Minimal code change (1 word!) - Each thread gets its own flag - No API changes - Works with existing code

Cons: - Still uses global state (just thread-local) - Doesn't help if single thread needs multiple concurrent bootstrap operations - C11 required (already using for errorString)

Complexity: □ Very Low **Recommended:** □ Yes, as first step

Option 2: Store in OneFile Structure

Change:

```
typedef struct {
    // ... existing fields ...
    bool isBootStrap;  // Add this field
} OneFile;
```

Update all code:

```
// Instead of: if (isBootStrap) ...
// Use:        if (vf->isBootStrap) ...

OneSchema *oneSchemaCreateFromFile (const char *filename) {
```

```
    // ...
    vf->isBootStrap = true;  // Set per-instance
    // ...
    vf->isBootStrap = false;
}
```

Pros: - No global state at all - Each OneFile tracks its own bootstrap state - More explicit and maintainable - Thread-safe by design

Cons: - Need to pass OneFile through all functions that check isBootStrap - Requires careful code audit to find all uses - More intrusive change

Complexity: □□ Medium **Recommended:** □ Yes, for thorough fix

---

Option 3: Bootstrap Context Structure

Change:

```
typedef struct {
    bool isBootStrap;
    // Could add other bootstrap-related state
    OneFile *tempFile;
    OneSchema *workingSchema;
} BootstrapContext;

OneSchema *oneSchemaCreateFromFile (const char *filename) {
    BootstrapContext ctx = { .isBootStrap = true };
    // Pass &ctx through function calls
}
```

Pros: - Encapsulates all bootstrap state - Extensible for future bootstrap needs - Thread-safe - Clear separation of concerns

Cons: - Need to thread context through multiple functions - Larger code change - Adds new structure to API

Complexity: □□□ High **Recommended:** ⚠ Overkill for this case

---

Option 4: Eliminate the Flag (Ideal but Complex)

Analysis: The flag is only used (commented out) in `oneReadLine` at line 1126:

```
// if (!isBootStrap) fprintf (stderr, "reading line %d type %c\n", (int)vf->line, t);
```

This is a debug statement that's already disabled!

Change:

```
// Option A: Just remove it entirely
// The flag serves no active purpose

// Option B: Make it a proper debug flag
#ifdef ONE_DEBUG
    if (vf->line > 0) fprintf(stderr, "reading line %d type %c\n", (int)vf->line, t);
#endif
```

Investigation needed: - Search for any other uses - Check if it was used historically for logic (not just debug) - Verify removing it doesn't break anything

Pros: - Removes the problem entirely - Simplifies code

Cons: - May have been needed for past logic - Need thorough testing

Complexity: □□□□ Unknown (needs investigation) **Recommended:** □ Investigate first

---

Recommended Approach

Phase 1: Quick Fix (5 minutes)

```
_Thread_local bool isBootStrap = false;
```

Why: Minimal change, fixes the immediate problem, uses C11 we're already using.

Phase 2: Proper Fix (1 hour)

```
typedef struct {
    // ... existing fields ...
    bool isBootStrap;  // Move here
} OneFile;
```

Then update: 1. `oneSchemaCreateFromFile`: Set `vf->isBootStrap` 2. Any functions checking `isBootStrap`: Change to check `vf->isBootStrap` 3. Remove global static

Why: Eliminates global state entirely, more maintainable.

Phase 3: Investigation (2 hours)

Determine if `isBootStrap` is actually needed:

```
# Check if it's used anywhere besides the commented line
grep -n "isBootStrap" ONElib.c
# Result: Only 4 lines - set true, set false, declare, commented debug
```

If only used for disabled debug statement → **remove it entirely**.

Implementation Details for Option 2

Step 1: Add field to OneFile

```
typedef struct {
    // Around line 145-150, add:
    bool isBootStrap;  // true during schema bootstrap

    // ... rest of struct ...
} OneFile;
```

Step 2: Update oneSchemaCreateFromFile

```
OneSchema *oneSchemaCreateFromFile (const char *filename) {
    // ...
```

```
    vf->isBootStrap = true;  // Line 292

    // ... all the bootstrap code ...

    vf->isBootStrap = false;  // Line 375
    return vs0;
}
```

Step 3: Update any checks

```
// In oneReadLine or elsewhere:
// Before:
// if (!isBootStrap) fprintf(...);

// After:
// if (!vf->isBootStrap) fprintf(...);
```

Step 4: Remove global

```
// Line 282: DELETE THIS LINE
// static bool isBootStrap = false;
```

Step 5: Test

```
# Rebuild
cargo clean && cargo build

# Test concurrent schema creation
cargo test test_concurrent_schema_from_text --test thread_safety_tests
cargo test test_concurrent_2_threads --test test_sequential_concurrent

# Should now work without mutex!
```

Testing the Fix

Before Fix

```
// This test FAILS without mutex
#[test]
fn test_concurrent_2_threads() {
    let h1 = thread::spawn(|| OneSchema::from_text("P 3 ta0\nO T 1 3 INT\n"));
    let h2 = thread::spawn(|| OneSchema::from_text("P 3 ta1\nO T 1 3 INT\n"));
    h1.join().unwrap();  // FATAL ERROR
    h2.join().unwrap();
}
```

After Fix

```
// This test PASSES without mutex
#[test]
fn test_concurrent_2_threads() {
    let h1 = thread::spawn(|| OneSchema::from_text("P 3 ta0\nO T 1 3 INT\n"));
```

```
    let h2 = thread::spawn(|| OneSchema::from_text("P 3 ta1\nO T 1 3 INT\n"));
    h1.join().unwrap();  //  OK
    h2.join().unwrap();  //  OK
}
```

Then Remove Rust Mutex

```
// src/schema.rs - DELETE THIS:
// static SCHEMA_CREATION_LOCK: Mutex<()> = Mutex::new(());
// let _guard = SCHEMA_CREATION_LOCK.lock().unwrap();

pub fn from_text(text: &str) -> Result<Self> {
    let c_text = CString::new(text)?;
    unsafe {
        let ptr = ffi::oneSchemaCreateFromText(c_text.as_ptr());
        // ... no mutex needed! ...
    }
}
```

Summary

Option	Complexity	Thread Safety	Recommendation
_Thread_local	□ Very Low	□ Yes	□ Quick fix
Store in OneFile	□□ Medium	□ Yes	□ Proper fix
Context struct	□□□ High	□ Yes	⚠ Overkill
Remove flag	□□□□ Unknown	□ Yes	□ Needs investigation

Immediate action: Use `_Thread_local` (1-line change) **Long-term action:** Move to `OneFile` struct (cleaner design) **Bonus investigation:** Can we remove it entirely?

The `_Thread_local` fix is so simple it's worth doing immediately, then the struct approach can be done as a proper refactoring for upstream contribution.