



GIOLIT | FORMAL VERIFICATION OF C PROGRAMS

Formal Verification Report: Errors Found

Document GIO-70036580E37D | issued 2026-10-03 01:25:07 IST (UTC+0530)

PROFESSIONAL LICENSE No. GIOLIT-PRO-2026-000011 licensed to **Acme Ltd.**

07_threads_race.c contains an error.

A concrete execution was found in which:

✗ **line 21:** the assertion fails

Proven for every input: it can never

✓ overflow a signed integer (1 place checked)

The failing execution is shown step by step on page 3 and can be replayed.

Program file	07_threads_race.c
Location	/tmp/gv-acme/07_threads_race.c
SHA-256	127133a8d4c9fe2031c82ed79bb4e7bd9f5a0a73c90d290836aea7ebfae34fc5
Size	23 lines, entry function main
Issued to	Giolit Labs
Machine ID	527d921416734c6985698c31556b62fc
Machine	Giolit Labs Verification — Ubuntu 24.04.5 LTS (kernel 7.0.14-orbstack-00380-ga7e0a2dc9535); VirtualApple @ 2.50GHz
Verified by	verifier@Giolit Labs Verification
Engine	Giolit Program Verifier v7 7.0.0, build e68a236-modified
Date and time	2026-10-03 01:25:07 IST (UTC+0530) (analysis 0.4 s)
Technique	Inductive invariants (abstract interpretation), Random interleaving
License	Professional license No. GIOLIT-PRO-2026-000011 , licensed to Acme Ltd.

How your client can check this certificate

- 1. Same file.** sha256sum 07_threads_race.c must print 127133a8d4c9fe2031c82ed79bb4e7bd9f5a0a73c90d290836aea7ebfae34fc5.
- 2. Licensed.** Issued under Giolit professional license No. GIOLIT-PRO-2026-000011, licensed to **Acme Ltd.**
- 3. Same result.** reproduce.sh re-runs the verification.
- 4. Ask us.** Write to business.giolitlabs@gmail.com quoting document GIO-70036580E37D.



GIOLIT — Formal Verification Services

Mathematical proof that your software is free of run-time errors.

We offer formal verification as a service: proofs of absence of run-time errors (overflow, division by zero, out-of-bounds access, invalid pointers, memory leaks) and of functional properties for C code, certificates for safety standards (DO-178C, ISO 26262, IEC 61508), and review of counterexamples with your engineers.

www.giolit.com

business.giolitlabs@gmail.com

www.giolit.com

The program

07_threads_race.c (SHA-256 127133a8d4c9fe20...):

```
1  /* Example 7 -- the same program without the mutex: a lost update.
2  * Expected: UNSAFE: both threads read x == 0 before either writes, so x == 1
3  * at the end. The counterexample is an interleaving of the two threads. */
4  #include <pthread.h>
5  extern void __assert_fail(const char *, const char *, unsigned int, const char *);
6  void reach_error(void) { __assert_fail("0", "07_threads_race.c", 5, "reach_error"); }
7
8  int x = 0;
9
10 void *worker(void *arg) {
11     x = x + 1;
12     return 0;
13 }
14
15 int main(void) {
16     pthread_t a, b;
17     pthread_create(&a, 0, worker, 0);
18     pthread_create(&b, 0, worker, 0);
19     pthread_join(a, 0);
20     pthread_join(b, 0);
21     if (x != 2) reach_error();
22     return 0;
23 }
```

Properties and results

#	Line	Property	CWE	Result	Method
1	11	Signed integer overflow: int32 overflow in +	CWE-190	✓ proven	Inductive invariants (abstract interpretation)
2	21	Assertion: assertion failure (reach_error reached)	CWE-617	✗ violated	Random interleaving

Verification technique

The program was compiled with the Clang C front end to LLVM IR and translated into a verification model: a control-flow graph whose transitions are exact linear relations over the program's variables, with machine integer ranges, wrap-around, pointers (object, offset), and memory allocation state. Every operation that can fail at run time becomes a *check*: an edge to an error location. A check is **proven** when no execution of the model reaches its error edge; it is **violated** when a concrete execution does. The model over-approximates the program (every real execution is an execution of the model), so a proof on the model is a proof for the program.

Techniques that decided this program:

- **Inductive invariants (abstract interpretation).** Inductive invariants were computed for every program point by abstract interpretation over template polyhedra (intervals, octagons and relational directions discovered from the program), with widening and narrowing. The invariant at the failing operation excludes the failure.
- **Random interleaving.** The threads were run on the exact program model in random orders, with random and boundary inputs; one interleaving violated the check. It respects thread creation, joins, atomic sections and locks, so it is a real execution.

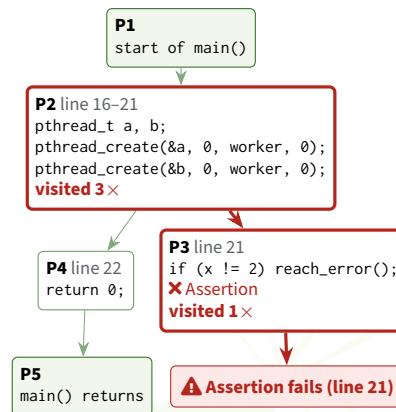
Counterexample 1: Assertion at line 21

Check: assertion failure (reach_error reached) (in main(), CWE-617).

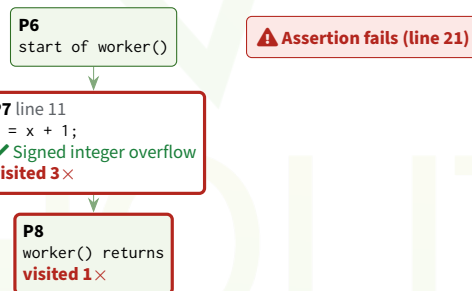
Found by: Random interleaving.

The failing execution on the control-flow graph. Red arrows are the transfers this execution takes (with how often); red boxes are the points it visits. The table after the graph lists every step with the values of the variables when execution arrives at the point (changed values in bold).

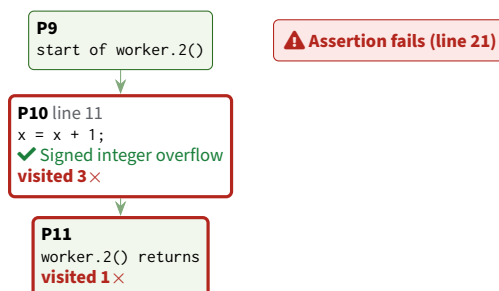
Function main()



Function worker()



Function worker.2()



The walk, step by step

Step	Point	Line	Source	Variable values on arrival
1	P2	16	<code>pthread_t a, b;</code>	<code>x = 0</code>
2	P7	11	<code>x = x + 1;</code>	<code>x = 0</code>

Step	Point	Line	Source	Variable values on arrival
3	P10	11	x = x + 1;	x = 0
4	P7	11	x = x + 1;	
5	P10	11	x = x + 1;	
6	P2	16	pthread_t a, b;	x = 0
7	P10	11	x = x + 1;	x = 1
8	P7	11	x = x + 1;	x = 1
9	P11		worker.2() returns	x = 1
10	P8		worker() returns	x = 1
11	P2	16	pthread_t a, b;	x = 1
12	P3	21	if (x != 2) reach_error();	
13	ERROR		the check fails	

Assumptions and scope

- **Inputs.** Values returned by functions without a body (for example `__VERIFIER_nondet_int`), the entry function's parameters and external data are arbitrary values of their C types: the result covers every input.
- **Machine model.** The host compiler's data model (x86-64, LP64: int 32 bits, long and pointers 64 bits), two's complement. Signed overflow is reported as an error; unsigned arithmetic wraps as C defines.
- **Entry point.** Executions start at `main()`. Only the source files listed on the first page are analysed.
- **Checks.** all kinds: assertions, signed overflow, division by zero, shifts, array bounds, pointer dereference, free, memory leaks.

Reproducing this result

The bundle delivered with this document contains:

- `07_threads_race.c` — the verified source, unchanged
- `certificate.json` — the statement of this result
- `results.json` — per-check results (machine-readable)
- `results.sarif` — SARIF 2.1.0 for code-scanning dashboards and CI
- `proof.json` — the verification model, the complete proof annotations and counterexamples
- `verifier-output.txt` — the verifier's console report
- `reproduce.sh` — the command that produced this result
- `certificate-password.txt` — the password that opens counterexample-report.pdf

Command (run in the bundle folder):

```
v7 07_threads_race.c
```

About Giolit

We offer formal verification as a service: proofs of absence of run-time errors (overflow, division by zero, out-of-bounds access, invalid pointers, memory leaks) and of functional properties for C code, certificates for safety standards (DO-178C, ISO 26262, IEC 61508), and review of counterexamples with your engineers. Visit www.giolit.com or write to business.giolitlabs@gmail.com.



Giolit — Formal Verification Services

Mathematical proof that your software is free of run-time errors.

We offer formal verification as a service: proofs of absence of run-time errors (overflow, division by zero, out-of-bounds access, invalid pointers, memory leaks) and of functional properties for C code, certificates for safety standards (DO-178C, ISO 26262, IEC 61508), and review of counterexamples with your engineers.

 www.giolit.com

 business.giolitlabs@gmail.com

