Exceptional Safety In Function Interfaces

Coding For The Non-Happy Path

Lou Langholtz 3/18/2023

Assumptions
Audience & Context

- You’re interested in software design.
- You’ve experienced software that failed partly to do an action.
- Desire/appreciate fault tolerance.
- Maybe unfamiliar with:
  - C++ “exception safety”.
  - Design by contract.
  - Atomicity.

Issue: Handling Errors At Function Interface

Context

- “Nobody ever spoke of 'error-safety' before C++ had exceptions” - David Abrahams 2000.
- ACID - 1983.
- Transaction concept - 1981.
- But still not widely spoken of?
    1. IBM MQ - “it’s very good at what it does”, “a top ranked solution”, “90% of the top 100 global banks”.
    2. RTI DDS - “the world leader - in DDS implementations”.

Issue: Handling Errors At Function Interface

Definitions

- Contract - collaboration in terms of obligations & benefits.
- Precondition - obligations, expected by function, else UB.
- Postcondition - benefits, expected after function returns.
- Error - when postcondition(s) can’t be met.
  - As output value.
  - *As exception, in supporting languages, like C++…*

Exception Is Thrown
What's the state?

- Consider C++ definitions we can fully look at like:

```
struct Context { vector<int> a; int* b; };

void setup(Context& context) {
  context.a.push_back(42);
  static int counter;
  ++counter;
  context.b = &counter;
}
```

- What's the state of `context` if `setup` throws an exception?

Exception Is Thrown

What's the state?

- Say we're dealing with a function like:

```
void setup(Context& context);
```

- What's the state of `context` if `setup` throws an exception?

Exception Is Thrown
What's the state?

- Say we're dealing with a function like:

```
void setup(Context& context);
```

- What's the state of `context` if `setup` throws an exception?

1. *Unchanged*.

Exception Is Thrown
What's the state?

- Say we're dealing with a function like:

```
void setup(Context& context);
```

- What's the state of `context` if `setup` throws an exception?

1. Unchanged.
2. *Completely setup.*

Exception Is Thrown

What's the state?

- Say we're dealing with a function like:

```
void setup(Context& context);
```

- What's the state of `context` if `setup` throws an exception?

1. Unchanged.
2. Completely setup.
3. *Anywhere in between!*

Exception Is Thrown

What's the state?

- Say we're dealing with a function like:

```
void setup(Context& context);
```

- What's the state of `context` if `setup` throws an exception?

1. Unchanged.
2. ~~*Completely setup*~~. Unlikely.
3. Anywhere in between!

Exception Is Thrown

What's the state?

- Say we're dealing with a function like:

```
void setup(Context& context);
```

- What's the state of `context` if `setup` throws an exception?

1. Unchanged.
2. ~~Completely setup~~. Unlikely.
3. *Anywhere in between!* State unknown. What could this mean?

Exception Is Thrown

State unknown…

Exception Is Thrown

State unknown…

- *Resources leaked?*

Exception Is Thrown

State unknown…

- Resources leaked?
- *Memory corrupted?*

Exception Is Thrown

State unknown…

- Resources leaked?
- Memory corrupted?
- *Invariants violated?*

Exception Is Thrown

State unknown…

- Resources leaked?
- Memory corrupted?
- Invariants violated?
- *Program in an invalid state?*

Exception Is Thrown

State unknown…

- Resources leaked?
- Memory corrupted?
- Invariants violated?
- Program in an invalid state?
- *Any possible: worst case in terms of exception safety!*

Exception Is Thrown

State unknown: anything possible…

- Has a name.

Exception Is Thrown

State unknown: anything possible…

- Has a name:
  - Wikipedia: *“No exception safety”*.

Exception Is Thrown

State unknown: anything possible…

- Has a name:
  - Wikipedia: “No exception safety”.
  - Cppreference: *“No exception guarantee”*.

Exception Is Thrown

State unknown…

- Resources leaked?
- Memory corrupted?
- Invariants violated?
- Program in an invalid state?
- *None possible: Second worst case in terms of exception safety!*

Exception Is Thrown

State unknown: but program has valid state etcetera…

- Has a name.

Exception Is Thrown

State unknown: but program has valid state etcetera…

- Has a name:
  - Wikipedia: *“basic exception safety”*.

Exception Is Thrown

State unknown: but program has valid state etcetera…

- Has a name:
  - Wikipedia: “basic exception safety”.
  - Cppreference: *“basic exception guarantee”*.

Exception Is Thrown

State unchanged…

Exception Is Thrown

State unchanged…

- Has a name.

Exception Is Thrown

State unchanged…

- Has a name.
  - Wikipedia: *“strong exception safety”*.

Exception Is Thrown

State unchanged…

- Has a name.
  - Wikipedia: “strong exception safety”.
  - Cppreference: *“strong exception guarantee”*.

Exception Is Thrown

What’s the state? Step back…

Exception Is Thrown

What's the state? Step back…

- *How about if the function signature instead is the following?*

```
void setup(Context& context) noexcept;
```

Exception Is Thrown

What's the state? Noexcept…

- How about if the function signature instead is the following?

```
void setup(Context& context) noexcept;
```

- *If exception actually thrown:* `std::terminate`.
  - Program done. Fails fast.
  - No ability to recover within program.
  - Lied to?

Exception Is Thrown

What's the state? Noexcept…

- How about if the function signature instead is the following?

```
void setup(Context& context) noexcept;
```

- *If exception actually thrown:* `std::terminate`.
  - Program done.
  - No ability to recover within program.
  - Lied to?
  - Non-starter, situation moot.

Exception Not Thrown

What's the state? Noexcept…

- How about if the function signature instead is the following?

```
void setup(Context& context) noexcept;
```

- *If actually non-throwing, no worries.*

Exception Not Thrown

What's the state? Noexcept…

- How about if the function signature instead is the following?

```
void setup(Context& context) noexcept;
```

- If actually non-throwing, no worries:
  - *Not an exception safety concern.*

Exception Not Thrown

What's the state? Noexcept…

- How about if the function signature instead is the following?

```
void setup(Context& context) noexcept;
```

- If actually non-throwing, no worries:
  - Not an exception safety concern.
  - *Can build on these functions.*

Exception Not Thrown

What's the state? Noexcept…

- How about if the function signature instead is the following?

```
void setup(Context& context) noexcept;
```

- If actually non-throwing, no worries:
  - Not an exception safety concern.
  - Can build on these functions.
  - *State as expected.*

Exception Not Thrown

State as expected: Noexcept…

- Has a name:
  - Wikipedia: *“No throw guarantee”*.

Exception Not Thrown

State as expected: Noexcept…

- Has a name:
  - Wikipedia: “No throw guarantee”.
  - Cppreference: *“Nothrow exception guarantee”*.

Issue: Handling Errors At Function Interface

Summary In Terms Of *Exceptions*

- Offer a category of assurance.
- Guarantees in decreasing order of safety:
  1. No-throw guarantee - for some C++ library ops.
  2. Strong exception guarantee - for key C++ library ops.
  3. Basic exception guarantee - for C++ library ops unless specified safer.
  4. None - everyone else's code?

Issue: Handling Errors At Function Interface

Summary In Terms Of *Errors In General*

- “Nobody ever spoke of 'error-safety' before C++ had exceptions” - David Abrahams 2000. ACID - 1983. Transaction concept - 1981. But still not widely spoken of?
- Offer a category of assurance.
- Safety categories in decreasing order:
    1. Function can’t error.
    2. State rolled back.
    3. No resources leaked, no invariants violated, state otherwise unspecified.
    4. State wholly unknown. Program may not even be in valid state. Most code?

More Example Code
What's the state?

- Consider C++ code like:

```
struct Context { vector<int> a; int* b; };

/// @pre <code>context.b</code> is <code>nullptr</code>.
/// @post <code>context.a</code> has 1 more element.
/// @post <code>context.b</code> points to a shared incremented counter.
/// @note If exception thrown, <code>context</code> is unchanged.
void setup(Context& context);
```

Issue: Handling Errors At Function Interface

My Conclusions

- APIs often aren’t designed for error mitigation.
- APIs often don’t specify how to recover when roll-back isn’t assured.
- We can categorize error situations and assurances.
- DBC can help.
- And you can too!