

A Practical Approach to Error Handling

- Errors can happen anywhere
- Want reliable program
- No time to write error handling

What do we do?

Options for Error Handling

think-cell 

```
file f("file.txt");
```

Options for Error Handling

```
file f("file.txt");
```

What happens if the file does not exist?

```
file f("file.txt");
```

What happens if the file does not exist?

- return value

```
file f;  
bool bOk=f.open("file.txt");  
if( !bOk ) {...}
```

- not for ctor

```
file f("file.txt");
```

What happens if the file does not exist?

- return value

```
file f;  
bool bOk=f.open("file.txt");  
if( !bOk ) {...}
```

- not for ctor

- out parameter

```
bool bOk;  
file f("text.txt",bOk);  
if( !bOk ) {...}
```

- clutter code with checks
 - can forget check - `[[nodiscard]]` for return values

Options for Error Handling (2)

- status: bad flag on first failure
 - single control path
 - good if checking at the very end is good enough
 - writing a file - ok
 - reading a file - maybe not
 - default for C++ iostreams

Options for Error Handling (2)

- status: bad flag on first failure
 - single control path
 - good if checking at the very end is good enough
 - writing a file - ok
 - reading a file - maybe not
 - default for C++ iostreams
- monad
 - goal: same code path for success and error case
 - like `std::variant<result, error>` + utilities
 - P0323R7 `std::expected`

Options for Error Handling: Exception

think-cell 

- exception

- exception
 - Catch exception objects always by reference
 - Slicing
 - Copying of exception may throw -> `std::terminate`

```
struct A {...};  
struct B : A {...};  
  
try {  
    throw B();  
} catch( A a ) { // B gets sliced and copied into a  
    ...  
    throw; // throws original B  
};
```

- exception
 - Catch exception objects always by reference
 - Slicing
 - Copying of exception may throw -> `std::terminate`

```
struct A {...};  
struct B : A {...};  
  
try {  
    throw B();  
} catch( A const& a ) { // B gets sliced and copied into a  
    ...  
    throw; // throws original B  
};
```

Options for Error Handling: Exception (2)

- work like multi-level return/goto
- add invisible code paths
 - one reason some code bases do not allow exceptions

```
auto inc(int i)->int { // throw(char const*)
    if(3==i) throw "Hello";
    return i+1;
}

auto main()->int {
    try {
        int n=3;
        inc(n); // throw(char const*)
        n=42;
    } catch( char const* psz ) {
        std::cout << psz;
    }
    return 0;
}
```

Options for Error Handling: Exception (2)

- work like multi-level return/goto
- add invisible code paths
 - one reason some code bases do not allow exceptions

```
auto inc(int i)->int { // throw(char const*)
    if(3==i) throw "Hello";
    return i+1;
}

auto main()->int {
    try {
        int n=3;
        inc(n); // throw(char const*)
        n=42;
    } catch( char const* psz ) {
        std::cout << psz;
    }
    return 0;
}
```

Options for Error Handling: Exception (3)

```
auto inc(int i, char const* & pszException )->int {
    {
        if(3==i) {
            pszException="Hello";
            goto exception;
        }
        return i+1;
    }
exception:
    return 0;
}
```

Options for Error Handling: Exception (4)

```
auto main()->int {  
    char const* pszException=nullptr;  
    {  
        int n=3;  
        inc(n,pszException);  
        if( pszException ) goto exception;  
        n=42;  
        return 0;  
    }  
exception:  
    {  
        std::cout << pszException;  
        return 0;  
    }  
}
```

Options for Error Handling: Exception (4)

```
auto main()->int {
    char const* pszException=nullptr;
    {
        int n=3;
        inc(n,pszException);
        if( pszException ) goto exception;
        n=42;
        return 0;
    }
exception:
    {
        std::cout << pszException;
        return 0;
    }
}
```

Stop whining! Of course must write exception-safe code!

Exception Safety Guarantees

(not really exception-specific)

Part of function specification

- Never Fails

(not really exception-specific)

Part of function specification

- Never Fails
- Strong Exception Guarantee:
 - may fail (throw), but will restore program state to what it was before: transactional
 - possible and desirable in library functions
 - very hard in application code
 - usually too many state changes

(not really exception-specific)

Part of function specification

- Never Fails
- Strong Exception Guarantee:
 - may fail (throw), but will restore program state to what it was before: transactional
 - possible and desirable in library functions
 - very hard in application code
 - usually too many state changes
- Basic Exception Guarantee:
 - may fail (throw), but will restore program to some valid state

Basic Exception Safety Guarantee

Customer: "Hello, is this Microsoft Word support? I was writing a book. Suddenly, Word deleted everything."

Microsoft: "Oh, that's ok. Word only provides a basic exception guarantee."

Customer: "Oh, alright then, thank you very much and have a good day!"

The Challenge

- Error handling is a lot of effort
 - in development
 - must be paranoid
 - create a lot of extra code
 - in testing
 - many codepaths to test
 - if you don't test them, they won't work

The Challenge

- Error handling is a lot of effort
 - in development
 - must be paranoid
 - create a lot of extra code
 - in testing
 - many codepaths to test
 - if you don't test them, they won't work
- Little customer gain

The Challenge

- Error handling is a lot of effort
 - in development
 - must be paranoid
 - create a lot of extra code
 - in testing
 - many codepaths to test
 - if you don't test them, they won't work
- Little customer gain
- So what do we do?

So what do we do?

- Check everything
 - check every API call
 - one wrapper per error reporting method
 - Windows: `GetLastError()`, `HRESULT`
 - Unix: `errno`
 - `assert` aggressively
 - asserts stay in Release
 - `noexcept` if caller does not handle exception
 - `std::terminate`, but unexpected exceptions will terminate anyway
 - install handler with `std::set_terminate` for checking

So what do we do?

- Check everything
 - check every API call
 - one wrapper per error reporting method
 - Windows: `GetLastError()`, `HRESULT`
 - Unix: `errno`
 - `assert` aggressively
 - asserts stay in Release
 - `noexcept` if caller does not handle exception
 - `std::terminate`, but unexpected exceptions will terminate anyway
 - install handler with `std::set_terminate` for checking
- Assume everything works

So what do we do?

- Check everything
 - check every API call
 - one wrapper per error reporting method
 - Windows: `GetLastError()`, `HRESULT`
 - Unix: `errno`
 - `assert` aggressively
 - asserts stay in Release
 - `noexcept` if caller does not handle exception
 - `std::terminate`, but unexpected exceptions will terminate anyway
 - install handler with `std::set_terminate` for checking
- Assume everything works
- Goal:
 - keep set of code paths small
 - keep set of program states small

If checks fail

- prio 1: collect as much information as possible
 - client: send core dump home
 - server: halt thread and notify operator

If checks fail

- prio 1: collect as much information as possible
 - client: send core dump home
 - server: halt thread and notify operator
- prio 2: carry on somehow
 - if check was critical, program behavior now undefined: no further reports
 - never terminate!
 - asserts can be wrong, too!
 - if you need safety (nuclear powerplant, etc.), add at higher level
 - example: server stops processing request categories with too many pending requests

Next: Homework

- Reproduce the error in the lab
- Add handling code only for errors that are reproducible
 - Otherwise you write
 - error handlers that are never used
 - error handlers that are never tested, do the wrong thing

5% of handlers handle 95% of errors

Next: Homework

- Reproduce the error in the lab
- Add handling code only for errors that are reproducible
 - Otherwise you write
 - error handlers that are never used
 - error handlers that are never tested, do the wrong thing

5% of handlers handle 95% of errors

- Not all errors equal

- `nullptr` access
- API calls not expected to fail (in this way)
- assertions

- `nullptr` access
- API calls not expected to fail (in this way)
- assertions
- "never happens"
 - no handler
 - like C++ undefined behavior: program is invalid

- `nullptr` access
- API calls not expected to fail (in this way)
- assertions
- "never happens"
 - no handler
 - like C++ undefined behavior: program is invalid
- Client: send report, disable future reports
- Server: send report, infinite loop (wait for debugger)
- Notify user only if false alarm unlikely
 - assertions may be wrong

```
auto RegisterFooHook(Foo foo) {  
    errcode_t err=RegisterFoo(foo);  
    if(err==SUCCESS) KeepTrackOfFoo(foo);  
    return err;  
}
```

- If `err` indicates error, do nothing

```
auto RegisterFooHook(Foo foo) {  
    errcode_t err=RegisterFoo(foo);  
    if(err==SUCCESS) KeepTrackOfFoo(foo);  
    return err;  
}
```

- If `err` indicates error, do nothing
- But no reproduction for `RegisterFoo` failing
- Effect on rest of the program?

```
auto RegisterFooHook(Foo foo) {  
    errcode_t err=RegisterFoo(foo);  
    if(err==SUCCESS) KeepTrackOfFoo(foo);  
    return err;  
}
```

- If `err` indicates error, do nothing
- But no reproduction for `RegisterFoo` failing
- Effect on rest of the program?
- Client: send report, throttle future similar reports
 - in Debug: notify developer
- Server: send report

Bad User Experience

- Reproducible 3rd party bug
 - sometimes PowerPoint makes shape disappear
- Reproducible, tested and supported

Bad User Experience

- Reproducible 3rd party bug
 - sometimes PowerPoint makes shape disappear
- Reproducible, tested and supported
- Not nice, users may complain

Bad User Experience

- Reproducible 3rd party bug
 - sometimes PowerPoint makes shape disappear
- Reproducible, tested and supported
- Not nice, users may complain
- Client/Server: only log, no report
 - to explain behavior if user calls

Indication of broken environment

- Other add-in hooked same function as us
- OS reports space as default decimal separator
 - both fully supported by us

Indication of broken environment

- Other add-in hooked same function as us
- OS reports space as default decimal separator
 - both fully supported by us
- Could still be cause of a problem

Indication of broken environment

- Other add-in hooked same function as us
- OS reports space as default decimal separator
 - both fully supported by us
- Could still be cause of a problem
- Client during remote support: notify support engineer
 - maybe reason for support call

- Reports with core dumps sent to server
 - automatically
 - if user opted out, user can send prepared email

- Reports with core dumps sent to server
 - automatically
 - if user opted out, user can send prepared email
- Error database
 - core dumps opened in debugger
 - errors automatically categorized by file/line
 - details and core dump accessible to devs

- Reports with core dumps sent to server
 - automatically
 - if user opted out, user can send prepared email
- Error database
 - core dumps opened in debugger
 - errors automatically categorized by file/line
 - details and core dump accessible to devs
- Devs can mark errors as fixed
 - trigger automatic update
 - or send automatic email - magic!

- Problem often related to customer environment
 - Proxy: list of loaded modules (DLLs, dylibs) in dump
- Can we identify module causing error?
 - or versions of module?

- Problem often related to customer environment
 - Proxy: list of loaded modules (DLLs, dylibs) in dump
- Can we identify module causing error?
 - or versions of module?

Report database with all reports

- 1 means has particular problem
- 0 means has different problem

0 1 1 0 0 1 0 1 0 1 1 0 (6 occurrences among 12 reports)

- Problem often related to customer environment
 - Proxy: list of loaded modules (DLLs, dylibs) in dump
- Can we identify module causing error?
 - or versions of module?

Report database with all reports

- 1 means has particular problem
- 0 means has different problem

0 1 1 0 0 1 0 1 0 1 1 0 (6 occurrences among 12 reports)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Problem often related to customer environment
 - Proxy: list of loaded modules (DLLs, dylibs) in dump
- Can we identify module causing error?
 - or versions of module?

Report database with all reports

- 1 means has particular problem
- 0 means has different problem

0 1 1 0 0 1 0 1 0 1 1 0 (6 occurrences among 12 reports)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Module B responsible? Or chance?

Minimum Description Length

- Compressing

0 1 1 0 0 1 0 1 0 1 1 0 (6/12)

- Knowing if reports contain module B helps compressing?

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

Minimum Description Length

- Compressing

0 1 1 0 0 1 0 1 0 1 1 0 (6/12)

- Knowing if reports contain module B helps compressing?

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- perfect arithmetic compression (Laplacian estimator)
 - estimates probability p that report has particular problem
- all p elem $[0,1]$ equally likely
- no. bits to compress N bits with K ones:

$\log [(N+1) * (N \text{ over } K)]$

- no. bits becomes smaller if p is closer to 0 or 1:
 - 12 bits with 6 ones: 13.55 bits
 - 12 bits with no ones: 3.70 bits

Compressing Reports

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

Compressing Reports

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits

Compressing Reports

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits
- Make use of module A
 - choose module A over B: 1 bit
 - compressing all reports with A (3/6): 7.13 bits
 - compressing all reports without A (3/6): 7.13 bits
 - total: 15.26 bits - module A has nothing to do with problem

Compressing Reports

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits
- Make use of module B
 - choose module B over A: 1 bit
 - compressing all reports with B (4/6): 6.71 bits
 - compressing all reports without B (2/6): 6.71 bits
 - total: 14.43 bits - still not relevant enough

Compressing Reports

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits
- Make use of module C
 - choose module C over A and B: 1.58 bits
 - compressing all reports with C (5/6): 5.39 bits
 - compressing all reports without C (1/6): 5.39 bits
 - total: 12.37 bits - relevant!

Compressing Reports

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits
- Make use of module C
 - choose module C over A and B: 1.58 bits
 - compressing all reports with C (5/6): 5.39 bits
 - compressing all reports without C (1/6): 5.39 bits
 - total: 12.37 bits - relevant!
- Extend to module versions
- More hypotheses make chance more likely

- new language feature
- `assert` on steroids
- declarative function pre- and postconditions

```
void push(int x, queue& q)
[[expects: !q.full()]]
[[ensures: !q.empty()]]
{
    ...
    [[assert: q.is_valid()]]
    ...
}
```

- When check contract?
 - debug
 - release
 - never
- What to do if contract violated?
 - terminate
 - carry on
 - report (what to whom?)

- When check contract?
 - debug
 - release
 - never
- What to do if contract violated?
 - terminate
 - carry on
 - report (what to whom?)
- removed from C++20 at last moment
- discussion will continue for C++23

THANK YOU!

think-cell 

for attending.

And yes, we are recruiting:

CPP-Summit

A Very Special Class of Errors

```
std::int32_t a=2 000 000 000;  
std::int32_t b=a+a;
```

What is **b**?

A Very Special Class of Errors

```
std::int32_t a=2 000 000 000;  
std::int32_t b=a+a;
```

What is **b**?

Uuh, may overflow.

- Let's check for it!

```
if( b<a ) {  
    ... treat overflow ...  
}
```

Ok?

Undefined Behavior (UB)

Example: int arithmetic overflow

A Very Special Class of Errors

```
std::int32_t a=2 000 000 000;  
std::int32_t b=a+a;
```

What is **b**?

Uuh, may overflow.

- Let's check for it!

```
if( b<a ) {  
    ... treat overflow ...  
}
```

Ok?

Undefined Behavior (UB)

Example: int arithmetic overflow

If program contains undefined behavior, compiler can do anything *with the whole program!*

- In particular, compiler may assume that UB never happens