think-cell

A Practical Approach to Error Handling

Introduction

think-cell

- Errors can happen everywhere
- 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

think-cell

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

What happens if the file does not exist?

Options for Error Handling

think-cell

```
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

Options for Error Handling

think-cell

```
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)

think-cell

- 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)

think-cell

- 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

Options for Error Handling: Exception

think-cell

- 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
};
```

Options for Error Handling: Exception

think-cell

- 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)

think-cell

- 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)

think-cell

- 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)

think-cell

```
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)

think-cell

```
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)

think-cell

```
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

think-cell

(not really exception-specific)

Part of function specification

- Never Fails

Exception Safety Guarantees

think-cell

(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

Exception Safety Guarantees

think-cell

(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

think-cell

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

think-cell

- 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

think-cell

- 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

think-cell

- 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?

think-cell

- 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?

think-cell

- 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?

think-cell

- 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

think-cell

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

If checks fail

think-cell

- 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

think-cell

- 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

Categories of errors

think-cell

```
immediate crash likely?        yes-->  Level 6
ex: imminent
    nullptr                            Client:
    access                             - error dialog
no                                       (false alarm unlikely,
|                                        increase chance of getting more
|                                        info)
|
|
v                                      and

program behavior               no--->  Level 5
well-defined?
ex: assert failed                      Client:
                                       - disable future reports
yes                                      (future behavior is ill-defined)
|
|                                      Server:
|                                      - infinite loop
|                                        (wait for debugger)
|
v                                      and
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
BORDER TO UNDEFINED BEHAVIOR LAND
vvv Programmer may have expectations of how program behaves vvv
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
situation has been           no--->  Level 4
tested?
ex: condition found                  Client or Server:
    that has never been              - send report
    reproduced
yes                                  Debug Build:
|                                    - error dialog (repro found!)
|
v                                    and

user experience is good      no--->  Level 3
ex: 3rd party
    bug we did                       Client or Server:
    not completely                   - log (explains behavior
    work around                        if we get complaints)
yes
|
|
|
v                                    and
```

```
situation may be indication  yes-->  Level 2
of broken program
environment                          Client during remote support session:
                                     - error dialog
no                                     (get attention of support eng)
|
|
|
v                                    and

situation occurs             no--->  Level 1
in every run/code path
                                     Client during remote support session:
yes                                  - or -
|                                    Debug:
|                                    - log (analyze to learn about
|                                      path to failure)
v

All good
```

Error Analysis

think-cell

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

Error Analysis

think-cell

- 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

Error Analysis

think-cell

- 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!

C++20 Contracts

think-cell

- 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()]]
...
}
```

C++20 Contracts (2)

think-cell

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

C++20 Contracts (2)

think-cell

- 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:

`hr@think-cell.com`

A Very Special Class of Errors

think-cell

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

What is `b`?

A Very Special Class of Errors

think-cell

```
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

think-cell

```
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